5-2016

The application of the Hadoop software framework in Bioinformatics programs

Dan Wang
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Dan Wang

Entitled
THE APPLICATION OF THE HADOOP SOFTWARE FRAMEWORK IN BIOINFORMATICS PROGRAMS

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

John A. Springer Chair	Dawn D. Laux
Eric Matson	
Kari L. Clase	
Michael A. Kane	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): John A. Springer

Approved by: Kathryne A. Newton 4/8/2016

Head of the Departmental Graduate Program Date

THE APPLICATION OF THE HADOOP SOFTWARE FRAMEWORK IN BIOINFORMATICS PROGRAMS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Dan Wang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2016

Purdue University

West Lafayette, Indiana

Dedicated to My Grandmothers.

ACKNOWLEDGMENTS

I wish to gratefully acknowledge my dissertation committee members Dr. Kari Clase, Dr. Dawn Laux, Dr. Eric Matson and Dr. Michael Kane for their insightful comments and guidance.

In addition, I would like to express my deepest gratitude to my advisor Dr. John Springer, who provided me with the sincerest believe, the steadiest support, and the most knowledgable guidance in the past three years.

Finally, I would like to thank my parents and my husband, who were always there cheering me up and stood by me through the good times and bad.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABBREVIATIONS

API	Application Program Interface
BLAST	Basic Local Alignment Search Tool
bp	Base Pair
cDNA	Complementary DNA
DNA	Deoxyribonucleic Acid
EC2	Elastic Compute Cloud
EST	Expressed Sequence Tag
GATK	The Genome Analysis Toolkit
Gb	Gigabyte
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
I/O	Input and Output
LSF	Platform Load Sharing Facility
m/z	Mass-to-Charge Ratio
Mb	Megabyte
MPI	Message Passing Interface
NCBI	National Center of Biotechnology Information
PCR	Polymerase Chain Reaction
RAM	Random-Access Memory
RNA	Ribonucleic Acid
SOAP	Short Oligonucleotide Analysis Package
SNP	Single Nucleotide Polymorphism

ABSTRACT

Wang, Dan Ph.D., Purdue University, May 2016. The Application of the Hadoop Software Framework in Bioinformatics Programs. Major Professor: John A. Springer.

The project described in this dissertation proposal attempted to improve the efficiency and scalability performance as well as the usability and user experience of three Bioinformatics applications – DNA/peptide sequence similarity comparison, digital DNA library subtraction, and DNA/peptide sequence de-duplication – by 1) adopting the Hadoop MapReduce algorithms and distributed file system and 2) implementing the fully automated Hadoop programs into a user friendly graphical user interface (GUI). In addition, the researcher was also interested in investigating the advantages and limitations of applying the Hadoop software framework as a general methodology in parallelizing Bioinformatics programs.

After considering the original calculation algorithms in the serial version of the programs, the available computational resources, the nature of the MapReduce framework, and the optimization of performance, a processing pipeline with one pre-processing step, three mappers, two reducers and one post-processing step was developed. Then a GUI interface that enabled users to specify input/output files and program parameters was created. Also implanted into the GUI were user friendly features such as organized instruction, detailed log files, multi-user accessibility, and so on.

The new and fully automated Hadoop Bioinformatics toolkit showed execution efficiency comparable with their MPI counterparts with median to large scale data, and better efficiency than MPI when ultra-large dataset was provided. In addition, good scalability was observed with testing dataset up to 20 Gb.

CHAPTER 1. INTRODUCTION

This chapter provides an overview of the presented study. In addition, the significance of the proposed software within the current canvas of Bioinformatics software development is established. Also presented here are the scope, research questions, assumptions, limitations and delimitations of the study.

1.1 Scope

The three Bioinformatics applications of interest included DNA/peptide sequence similarity comparison (also called Basic Local Alignment Search Tool or BLAST) (S. F. Altschul, Gish, Miller, Myers, & Lipman, 1990), digital DNA sequence/library subtraction (Kane et al., 2008), and DNA/peptide sequence de-duplication. Specifically, the BLAST tool maps DNA or peptide sequences against known or self-defined libraries, and returns sequences from the libraries that match the query sequences with high similarities and high confidence levels (S. F. Altschul et al., 1990); digital DNA sequence/library subtraction calculates the similarities of cDNAs from two similar libraries/genomes, and identifies the unique sequences that are only expressed in one library/genome but not the other (Kane et al., 2008); DNA/Peptide sequence de-duplication compares two given sets of sequences, removes the same or similar sequences between the two based on either the defined confidence level or percentage similarity and returns one combined set of sequences.

At the first glance, the above three Bioinformatics programs of interest seemed unrelated; however, a closer investigation of the data analysis and transformations conducted in these applications revealed the relations among them. As shown in Figure 1.1, DNA/peptide sequence similarity comparison identifies the

overlapped sequences between two libraries; digital DNA sequence/library subtraction focuses on the non-overlapped sequences; DNA/Peptide sequence de-duplication removes one copy of the overlapped sequences and combines everything else.

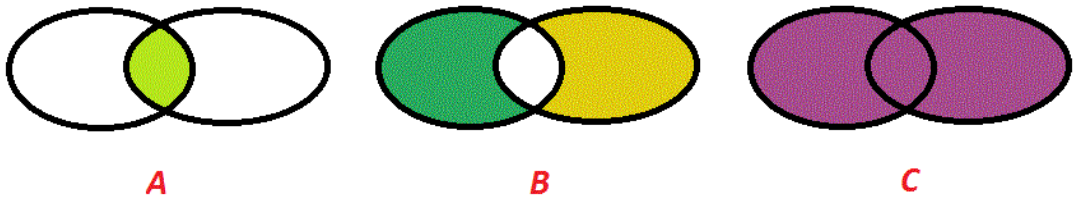

Figure 1.1. The Venn diagram of the three Bioinformatics applications of interest.

A: DNA/peptide sequence similarity comparison

B: Digital DNA sequence/library subtraction

C: DNA/Peptide sequence de-duplication

The above description and Figure 1.1 provide a high level and over-simplified summary of the functionalities of the three programs of interest. The complications brought by the biological system and the modern sequencing techniques will be presented in Section 2.3.1.

The original serial programs of the above applications work well with small genomes, or small input datasets; however, with larger input datasets, more scalable and reliable programs are needed. As a result, the researcher proposed here to build the distributed Hadoop versions of the above programs. Although the parallel version of a similar DNA sequence comparison and search program, mpiBLAST, already exists (Darling, Carey, & Feng, 2003), the researcher believed that the proposed Hadoop version had multiple potential advantages, which are discussed in Section 2.2.1.

It should also be noted that mpiBLAST was designed for genome comparisons, while the proposed Hadoop Digital DNA Library Subtraction tool focuses more on identifying the subset of different sequences between two similar or biological related genomes or cDNA libraries, and the sequence de-duplication tool aims to remove the duplicates, or sequences with high similarities between two given libraries, to facilitate easier downstream sequence manipulations. Similar to the original BLAST program, the three proposed Hadoop programs utilize dynamic programming algorithms (Pearson & Miller, 1992) to optimize local sequence alignment. The Hadoop MapReduce algorithm and Distributed File System (HDFS) (Dean & Ghemawat, 2008) were utilized to benefit the programs from both the efficiency and the scalability perspectives. In order to determine the reliability and the performance of the new programs, artificial as well as real sequence files from different origins with known variations and various sizes were analyzed using the new Hadoop version programs that the researcher developed. As comparisons, same calculates were also carried out using the original serial programs (or parallel programs, where applicable). The calculation results and performance data were documented, and the comparisons of the results are discussed in Chapter 4 and Chapter 5.

1.2 Significance

The significance and novelty of the proposed work was two-fold. First of all, the three Bioinformatics applications discussed above are widely involved in various scientific research areas. Currently, digital DNA sequence/library subtraction is performed typically by using BLAST; however, as mentioned above, BLAST is primarily a sequence alignment and comparison tool. In order to achieve library subtraction purposes, complex data preprocessing, several rounds of BLAST, and BLAST result sorting and reformatting are needed. The automation of the whole process is likely to benefit scientific research from various disciplines. For example,

the identification of unique DNA/RNA sequences from patient genomes of a specific disease versus genomes from healthy individuals would potentially result in the identification of disease causing genes; the comparison of sequences from orthologue and/or paralogue tissues and organs might reveal significant phylogenomic evidences (Koonin, 2005).

DNA/peptide de-duplication is a sequence manipulation step that has been widely used in a large range of Bioinformatics tasks. For example, the removal of overlapped exons, the compiling of genomes from different sources, and so on. However, sequence de-duplication is a time-consuming task. A bit to bit exact comparison takes up to quartic time ($O(n^4)$) to process. Even with the help of the BLAST tool, a search with user defined similarity level takes up to cubic time ($O(n^3)$) to complete. As a result, although DNA/peptide de-duplication is a conceptually very straightforward task, it often limits the effective analysis of genomic/proteomic data. On the other hand, the computational intensive calculations in sequence de-duplication tasks could easily be decomposed into parallel queries and the combination of query results, indicating that the adoption of the Hadoop MapReduce algorithm could effectively speedup as well as automate the sequence de-duplication operation.

The second fold novelty of the proposed work was that the transformation of Bioinformatics programs to Hadoop software framework is an important methodology to study (Leo, Santoni, & Zanetti, 2009). Currently, a few popular Bioinformatics programs, such as BLAST (Leo et al., 2009), Genome Analysis Toolkit (GATK) (McKenna et al., 2010) and Bowtie (Niemenmaa et al., 2012) already have Hadoop versions, but most of these previous works were developed based on existing MPI codes and failed to take full advantages of the critical characteristics of the MapReduce concepts (O'Driscoll, Daugelaite, & Sleator, 2013). It had been realized that Hadoop had its own limitations, which are discussed in Section 2.2.2, and not all programs were compatible with the notion of MapReduce and HDFS. However, the researcher believed that most Bioinformatics

applications would benefit from converting to Hadoop and utilizing MapReduce algorithms based on the following reasons:

- Bioinformatics programs are typically computational intensive and required heavy I/O. As a result, computation and I/O parallelization strategies that scale well with data size are in demand.

- Most Bioinformatics programs were written by biological researchers who are not experts in complicated parallelization strategies (Taylor, 2010). MapReduce algorithms were reported to present lower programming complexities than other parallelization strategies. The reasons includes: 1) all low level details, such as exception handling, data partitioning, and task/job tracking, are handled automatically by the framework itself, and the programmers only need to develop and optimize mappers and/or reducers (White, 2012); and 2) a large range of programming languages could be used to created mappers and reducers using the Hadoop Streaming API (White, 2012).

- Hadoop required that input data could be split into independent blocks with sizes up to the size of the computer node RAM. Within this constraint, Hadoop provided scalable and reliable environment for parallel computation. (White, 2012) As a result, most Bioinformatics applications are well suited for the MapReduce paradigm.

1.3 Research Question

Based on the researcher's pilot study, the proposed research attempted to answer the following questions:

- Can Hadoop's MapReduce algorithm and Distributed File System be utilized to build Bioinformatics applications that automate the following processes: 1)

DNA/peptide sequence similarity comparison; 2) digital DNA sequence/library subtraction; and 3) DNA/peptide sequence de-duplication?

- Compared to the serial or other parallelization strategies, could the above methods improve the efficiency and scalability of the programs of interest?
- Could Hadoop be used as a general parallelization strategy to optimize the performance of other Bioinformatics applications?

1.4 Limitations

The research described in this dissertation was conducted under the following limitations:

- To test the scalability of the developed programs, large artificial datasets with known variances were used. Although those variances did not occur naturally, the researcher incorporated representative frequencies and variance densities when designing them.
- Because Hathi (hostname: hathi.rcac.purdue.edu, website: https://www.rcac.purdue.edu/compute/hathi/) was the only Hadoop cluster to which the researcher had access, all Hadoop related experiments and testing described in Section 4.3 were executed in the Hathi cluster. All cluster computers are different, and the performance of the codes in other Hadoop instances could be different.
- All experiments described in Section 4.3 that did not require Hadoop, for example, the testings related to the execution of the serial or MPI versions of the BLAST tool were carried out in Conte (hostname: conte.rcac.purdue.edu, website: https://www.rcac.purdue.edu/compute/conte/). Similarly, the performance of the codes in other cluster systems could be different.

1.5 Assumptions

The project was designed based on the following assumptions:

- The calculation algorithms from the existing Bioinformatics programs were used in the new Hadoop mappers and reducers. The researcher assumed that those algorithms were correct.
- It was assumed that with the same calculation algorithms, the adoption of the MapReduce parallelization strategy would not change the calculation accuracy, or the differences were negligible. Experiments described in Section 3.5 intended to prove this.
- It was assumed that Hathi and Conte, the computer clusters that were used for testing purposes in this project, had been fully tested and optimally configured to run Hadoop and other jobs, and that their performance was representative of all distributed clusters. The configurations of Hathi and Conte are introduced in Section 3.3.
- Specific computer hardware and software used to implement solutions did not alter the generalization of the results unless specifically mentioned.
- In Section 4.3, the execution times of jobs conducted in Hathi and in Conte were compared directly. It was assumed that the single node performance of the two clusters were comparable, if not the same.

1.6 Delimitations

The following delimitations were acknowledged:

- The proposed research focused on improving the efficiency and scalability by the implementation of Hadoop's MapReduce algorithm and Hadoop Distributed File System (HDFS), but not the underlying calculation

algorithms. By optimizing the calculation algorithms, the researcher could also improve the performance of the programs, but this was beyond the scope of the current project.

- Two types of databases, the NCBI database (National Center for Biotechnology Information database, which is one of the most commonly used database), and database generated from input sequence files, were incorporated into the proposed programs. The choice to use alternative database would be possible but not optimized.

- Data transfer time, which could be significant if files are large, was not included when efficiency and scalability performance were discussed and compared. Examples of data transfer includes data downloading from Internet source to local file system, data replication between different cluster systems, and so on, but not includes the transfer of files between HDFS and local file systems.

- Although numerous algorithms are available, in the Digital DNA Library Subtraction tool or the Sequence De-duplication tool, none of the methods other than the one mentioned above were considered.

- When the usability features in the graphical user interface were designed, only the needs of academic and professional users were taken into consideration.

1.7 Summary

This chapter presented an overview of the research project and introduced the motivation, scope, significances as well as the research questions of the proposed research. A list of limitations, delimitations and assumptions of the chosen scope has also been noted.

In the next chapter, brief introductions of Bioinformatics, the scientific applications of sequence similarity comparison, DNA library subtractions and

sequence de-duplication are presented. Also included are the advantages and limitations of using Hadoop software framework in Bioinformatics applications.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

In this section, brief introductions of Bioinformatics, the scientific applications of sequence similarity comparison, DNA library subtractions and sequence de-duplication, and the advantages and limitations of the Hadoop software framework are presented. A discussion of the recent efforts in improving the efficiency and scalability of Bioinformatics software by using Hadoop methodology as well as motivations of the proposed project are also addressed.

2.1 Bioinformatics and Bioinformatics Software

Bioinformatics is an interdisciplinary field that has arisen from the demand of biologists and medical scientists to take advantages of modern computer power and technology to preserve, organize, annotate, interpret, and analyze the massive amount of data that is constantly being produced in genomic and proteomics studies (Akalin, 2006; Sugano, 2009). Bioinformatics could be loosely defined as the application of information technology and related theories to the field of biology, especially genomics and molecular biology (Cohen, 2004), with the ultimate goal of developing automate computational models that complement *in vivo* and *in vitro* biological experiments (Rothberg, Merriman, & Higgs, 2012) and related medical operations (Chen, He, Zhu, Shi, & Wang, 2015). Figure 2.1 illustrates a typical Bioinformatics research workflow.

Historically, the bottleneck that limited the development of Bioinformatics was the cost of obtaining sufficient and reliable biological data. The Human Genome Project completed the sequencing of the first human genome in 2003, at the cost of approximately 2.7 billion US dollars in a two-year period (NIH-National-Human-Genome-Research-Institue, 2015). Currently, with the

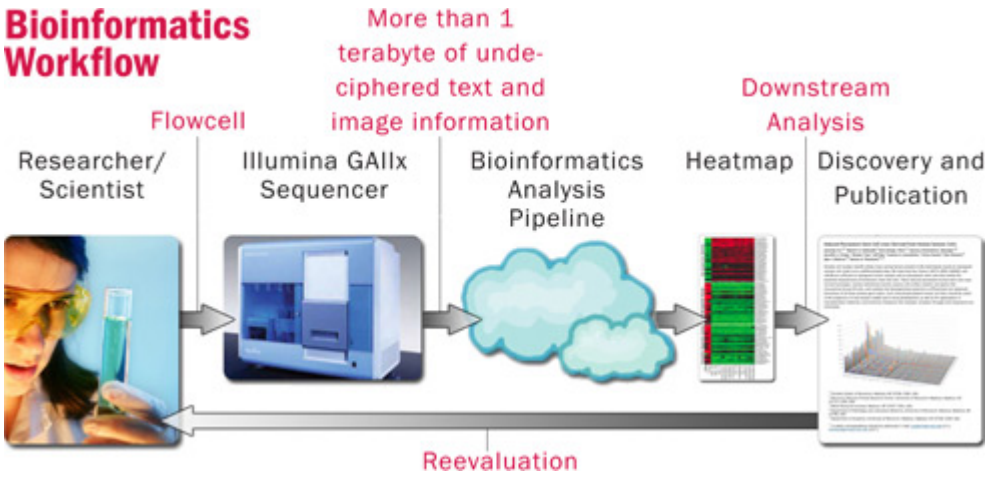

Figure 2.1. A Bioinformatics Workflow.

(Downloaded from `https://morgridge.org/bioinformatics-2/`)

booming of new technologies, such as various next generation sequencing techniques (Bentley et al., 2008; Drmanac et al., 2010; Margulies et al., 2005; McKernan et al., 2009), high resolution mass spectrometry techniques (Liu et al., 2014), high throughput DNA and RNA microarray techniques (Schena, Shalon, Davis, & Brown, 1995) and so on, the cost of money and time for obtaining raw data has become lower and lower. Alternatively, with the dropping of expenses, the scale of Bioinformatics projects are constantly growing, which has resulted in biological datasets with larger and larger sizes. For example, the 1000 Genome project alone generated almost five terabytes of raw data (Pennisi, 2010). As a result, in order to process the data, more and more Bioinformatics programs have been developed to automate the data analysis process, and various parallelization strategies have been considered to improve their performance.

Due to the interdisciplinary nature of Bioinformatics, current Bioinformatics programs were either developed by computer programmers who collaborated with biological researchers, or biologists who had some levels of programming background and wanted to customize their specific scientific needs. The latter situation is

becoming more and more prevalent, because it normally takes vast amounts of effort and time to explain the sophisticated scientific principles behind the desired program to computer programmers who have no prior biological research backgrounds, and make sure that the developed program does exactly what the researchers envisioned it to do (Macaulay et al., 2009).

The primary and original consideration of most Bioinformatics programs was accuracy. Researchers cared more about the precision and confidence of the computing results rather than how much time it took the program to process the input data. Under this scenario, the serial implementations of programs developed by biological researchers themselves worked pretty well, because they were capable of providing the confidence and flexibility that most researchers required. However, with the exponential growth of the size of data that needs to be processed, efficiency and scalability are becoming significant concerns, which created a difficulty for biologists to develop customized programs as most researchers lack the experience and expertise to manipulate large datasets and optimize sophisticated traditional parallelization strategies such as MPI (Message Passing Interface) (Gorlatch & Bischof, 1998) or OpenMP (Gabriel et al., 2004).

Under the above circumstances, the Hadoop software framework (Shvachko, Kuang, Radia, & Chansler, 2010), which will be introduced in great depth in the next section, provided the Bioinformatics researchers with a general purpose parallelization strategy with relatively low programming complexity and straightforward parallelization concepts.

2.2 Hadoop: Advantages and Disadvantages

This section discusses generally the advantages as well as the limitations of the Hadoop software framework.

2.2.1 Advantages of Hadoop

The application of the Hadoop framework in the parallelization of Bioinformatics programs was motivated by the following merits:

- Hadoop has low program complexity (Nordberg, Bhatia, Wang, & Wang, 2013).

  Under the Hadoop framework, programs are expressed as a series of map and reduce operations conducted on independent and replicated chunks of input data. Although some programs cannot be easily expressed in this way, programs compatible with this partition strategy usually benefit from services provided by the Hadoop framework. For example, the Hadoop framework automatically handles all low level parallelization details such as data partitioning, data loading, job tracking, task scheduling, data sorting and routing among processors, computer node failure handling and so on. In addition, users are allowed to use a large range of programming languages (not only Java) such as C++, Python and Perl. through a generic API called Hadoop Streaming to create mappers and/or reducers (White, 2012).

- Hadoop utilizes a unique distributed file system called HDFS (Chang et al., 2008), which brings computation to the same or the nearest computer node where the data was preserved. On the other hand, traditionally in parallel computing paradigms, data partitions are arbitrarily sent to computation resources. The application of HDFS was aimed at minimizing intra-node data transfer, and hence improving the overall computation performance (Taylor, 2010).

- Hadoop provides a robust, reliable and fault-tolerant file and computation system.

  Typically, in the Hadoop framework, data is split into independent chunks and each chunk is replicated multiple times across different computer nodes. Under Hadoop's share-nothing architecture, computations that operate on

each chunk of the data must be independent of each other in both the map and the reduce phase. The only exception happens when the outcome from mappers are fed into reducers. Furthermore, the Hadoop framework comes with job and task trackers, which monitor computer node or computing task failures and control the restart of threads with replicated data from other healthy and standby node. In other words, the design of HDFS file system guarantees that a single point of failure does not exist (White, 2012).

- With all the above being said, good scalability is probably the most important feature of Hadoop. This characteristic has made Hadoop a unique and promising software platform. In the implementation of Bioinformatics applications, the near linear scalability of Hadoop makes it superior to other parallelization strategies (Schumacher et al., 2014). Most Bioinformatics applications were developed and tested based on relatively small datasets. However, with the rapid development of new biological and biochemical technologies, such as various next generation sequencing techniques, proteomic techniques, and high throughput screening techniques, the analysis of ultra-large-scale datasets with Bioinformatics programs is in demand and becoming the bottleneck that hinders the generation of new knowledge. Traditional programs often fail to scale with the size of input data by demanding too much memory, and/or taking polynomial time. Hadoop programs, on the other hand, were reported to scale automatically and linearly with datasets up to the petabytes range as had been demonstrated by multiple recent publications (Langmead, Schatz, Lin, Pop, & Salzberg, 2009; Schumacher et al., 2014).

2.2.2 Limitations of Hadoop

Just like any other methodology, the Hadoop software framework presents various advantages, but it is not a cure-all. The researcher also observed the following limitations of this framework.

- The MapReduce algorithm is not suitable to solve every scientific problem.

  The general idea of parallelization is to divide a data analysis process into multiple independent sub-processes, and distribute them over multiple processors to obtain greater efficiency. However, one should realize that not all data analysis processes can be split into independent pieces. For example, in a Bioinformatics scenario, the *de novo* assembly of short DNA reads, which come from genome sequencing, involves large graphic processing (Iqbal, Caccamo, Turner, Flicek, & McVean, 2012). In other words, the entire input dataset needs to be considered together at the same time, and thus cannot be split into independent pieces. As a result, although *de novo* assembly involves a huge amount of input data and heavy computation, serial codes are still used as the prevalent algorithms to ensure accuracy. In some other multi-step designs, although some of the steps can be easily and safely split, parallelization strategy are not optimal for the speed-limiting step. Under such circumstances, the application of Hadoop is not able to greatly improve the analysis efficiency (Taylor, 2010).

- Hadoop is a open source and free application, but in order to obtain optimal performance, it requires designated clusters, as well as highly experienced personnel to design and maintain (White, 2012). This arises cost challenges for its full-scale usage.

- The total execution time is the sum of computation time, communication time and I/O time (Schikuta & Wanek, 2001). For Hadoop programs, because tasks are separated into map phases and reduce phases, and the input data are split into data chucks that have multiple copies, the I/O demands are

extremely heavy. Although parallel I/O strategies have been implemented into later versions of the Hadoop framework, the I/O loads still have high requirements for computer hardware (Shvachko et al., 2010).

- Compared with hand crafted MPI parallelization, the Hadoop version of the same program is normally less efficient for the following reasons.
  - Unlike their MPI counterparts, the MapReduce parallelization strategy is generic and not optimized for each individual situation and scientific problem (which is also why the programming complexity is low in Hadoop).
  - Hadoop's initialization latency (Nordberg et al., 2013) is much higher than MPI. In other words, efficiency-wise, MPI solutions are more optimized for small to median scale data processing, and Hadoop's advantage is its horizontal scalability in large scale data.

2.3 Sequence Similarity Comparison and Digital DNA Library Subtraction

In previous sections, the researcher established the reasons that Hadoop was a valuable software framework to parallelize various Bioinformatics programs and improve their performance. Also revealed were some of the limitations. In this section, the biological meaning and importance of sequence similarity comparison and digital DNA library subtraction are discussed.

2.3.1 Sequence Similarity Comparison and BLAST

DNA molecules consist of different combinations and permutations of four kinds of nucleotide bases (A, T, G, C) building blocks. Similarly, the basic units of peptide molecules are 20 different amino acids building blocks(A, C, D, E, F, G, H,

I, K, L, M, N, P, Q, R, S, T, V, W, Y). The above abbreviations were explained in Table 2.1.

Table 2.1
The Abbreviations of Nucleotide Bases and Amino Acids Building Blocks.

Nucleotides			
A	Adenine	T	Thymine
G	Guanine	C	Cytosine
Amino Acids			
A	Alanine	C	Cysteine
D	Aspartic Acid	E	Glutamic Acid
F	Phenylalanine	G	Glycine
H	Histidine	I	Isoleucine
K	Lysine	L	Leucine
M	Methionine	N	Asparagine
P	Proline	Q	Glutamine
R	Arginine	S	Serine
T	Threonine	V	Valine
W	Tryptophan	Y	Tyrosine

Generally speaking, the alignments or mappings of DNA or peptide sequences are just the comparison of strings that contain those basic units (four nucleotide bases in the case of DNA and 20 amino acids in the case of peptides), which represents a lot of similarities to other general purpose string comparisons and information search tasks; however, the following uniquenesses have to be taken into considerations:

- The wide existence of synonymous units.

The biological function of DNA molecules is to store genetic information that could be transcribed into RNA and then translated into protein molecules. The biological functions of protein molecules include catalyzing biochemical reactions, responding to attacks from foreign particles such as bacteria and virus, transporting nutrients to proper destinations, and so on. The amino acid sequence of a protein/peptide molecule is called the 2-D structure; however, the biological functions mentioned above rely on not only the 2-D structure, but also the maintenance of proper 3-D spacial structures of protein molecules. In other words, rather than the sequence strings themselves, the peptide sequence that encoded by a DNA molecule, or the 3-D structure that preserved by a protein/peptide molecule, are the things that one should care. Figure 2.2 illustrates simplified relationships between DNA, RNA, amino acid sequence, and the 3-D structure of a protein.

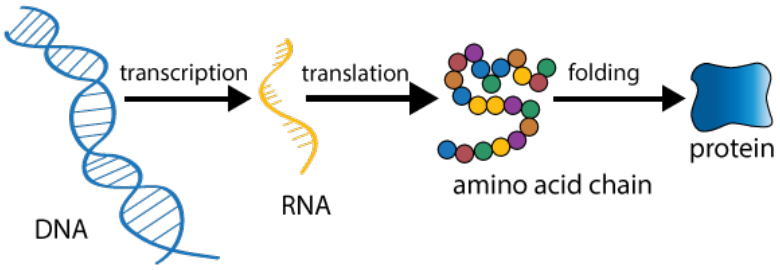

Figure 2.2. The translation of DNA and the folding of protein 3-D structure.

(Source: `http://biosocialmethods.isr.umich.edu/wp-content/uploads/2014/09/central-dogma-enhanced.png`)

Along the stream of evolution, in order to protect the robustness of genetic information, decrease the risk of genetic mutations, while keeping the accuracy and sensitivity of heredity, synonymous genetic codons, which means different DNA combinations translated to the same amino acid unit, have been developed by the nature. For example, both the combination of nucleotides AAA and AAG translates into amino acid L (Lysine). Generally speaking, the

third letter of a DNA codon does not matter as much as the first and the second positions.

In the case of protein molecules, slightly different sequences could be constructed into either very similar (in the case of benign mutation), or highly different (in the case of disease causing mutations) 3-D spacial structures, depending on the nearby chemical and biological conditions. To make things even more complicated, the harmful level of non-synonymous mutations are not all the same. Fatal mutations are very rarely observed, light to median level disease causing mutations are more common, while totally benign mutations have also been widely reported.

In other words, the alignment tools that compare DNA and peptide molecules have to take all the above factors into consideration and penalize differently between synonymous and non-synonymous mutations, and between non-synonymous mutations with different harmful levels.

- The uniqueness brought by the next generation sequencing techniques.

  Both DNA and protein molecules are macromolecules. A natural DNA molecule could be anywhere between 500 base pair to 10^9 base pair long, and the average size of a human DNA is about $5 * 10^8$ base pairs. A protein molecule could consist up to 30,000 amino acid building blocks. In order to analyze the sequences and spacial structures of those macromolecules, what people normally do is to physically or enzymatically break them down into shorter pieces, sequence the latter, and assemble the fractions back to large molecules based on purposefully left overlapped portions and artificial molecular handles. In order to obtain accurate comparison and alignment results with high confidence, those overlapped portions and handles have to be treated differently.(Quail et al., 2012)

- The calculation of alignment confidence level.

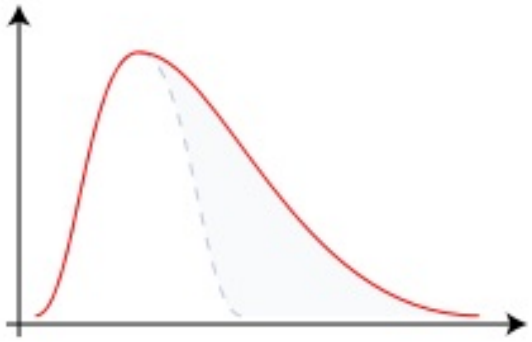

Figure 2.3. A right skewed bell curve that represented the distribution of alignment scores.

When a sequence is compared to a known library of sequences to determine the best match, it has to be kept in mind that the alignment scores are not normally distributed. Because only the best alignment score is considered for each query sequence, the distribution is highly left skewed with heavy right tail. In other words, if the critical value is calculated based on a normal distribution, the confidence level would likely to be over estimated. Figure 2.3 shows an example of such a skewed distribution.

BLAST is by far one of the most popular and successful Bioinformatics applications that has been developed to compare the similarity levels of DNA/RNA or protein/peptide sequences (Madden, 2013). The following characteristics (S. F. Altschul et al., 1990) of the traditional NCBI BLAST tool ensure both the accuracy and the efficiency of the sequence comparison and query:

- BLAST encodes all known synonymous DNA and protein mutations into the alignment algorithm.
- Nucleotide bases and amino acids are further divided into subgroups based on their properties, and the non-synonymous mutations within each subgroup received less penalties than across subgroup mutations. This is a heuristic

calculation and sometime maybe incorrect, however, in practice, this is by far one of the best approximation strategy that balanced the computation complexity and accuracy.

- A library of "special" sequences, which includes widely used sequencing handles and commonly seen repeating fractions is provided, so that the above short components are treated differently.
- The distribution of the alignment scores are approximated by using Monte Carlo resampling method (Fishman, 2013), so that the confidence score could be calculated unbiasedly regardless of the underlying distribution.
- A dynamic programming approach is conducted to guarantee the optimal solution as well as the efficiency of the alignment operation.
- Commonly used DNA and protein libraries, such as Ecoli, human, rat, drosophila, maze, and so on, are pre-loaded to the memory of the data server, which ensures the speed of online queries.
- To further improve the accuracy and efficiency of the BLAST tool, modified algorithms, such as DELTA-BLAST (Boratyn et al., 2012), gapped and PSI-BLAST (S. Altschul et al., 1997), mpiBLAST (Darling et al., 2003) and so on, have been developed and reported by a broad range of researchers.

2.3.2 Digital DNA Library Subtraction

DNA library subtraction is a process where the DNA libraries of two similar genomes are compared, and the sequences that are uniquely expressed in one genome, but not the other, are identified (Kane et al., 2008). In downstream analysis, the identified DNA sequences are usually confirmed by DNA microarray experiments (Heller, 2002), and possible genetic functions of the true hits are further annotated by comparing to a database of genes or proteins with known

functions (Thomas, Wood, Mungall, Lewis, & Blake, 2012). Historically, DNA library subtraction was typically done by conducting PCR (polymerase chain reaction) experiments (Diatchenko et al., 1996). In the last decade, the PCR based DNA library subtraction method was widely used to identify disease causing genes or differentially expressed genes in human cancer (Houghton et al., 2001; Jiang et al., 2002; Xu et al., 2000), but gradually replaced by digital library search after the mature of Bioinformatics software BLAST (S. F. Altschul et al., 1990). However, as a general purpose DNA, RNA and peptide database searching and alignment tool, BLAST is not specifically designed for library subtraction tasks, and thus tedious data pre-processing and post-processing steps are needed, which could be as time consuming as the search step itself. As a result, a more efficient software that is specifically designed for and have the capability of automating the whole process of digital DNA library subtraction and downstream analysis steps is in demand, especially when large datasets are to be analyzed.

2.4 Previous Attempts in Applying Hadoop in Bioinformatics Software

Many research groups had realized the invaluable role the Hadoop software framework could play in the development and optimization of Bioinformatics applications. Currently, there were more than 20 published works that focused their efforts on moving existent Bioinformatics software to Hadoop implementations (O'Driscoll et al., 2013); however, that still only represented a tiny portion of all Bioinformatics applications. By reviewing some of those previous achievements, the researcher would like to demonstrate not only the great potential of the Hadoop framework in the design and development of Bioinformatics programs, but also a need of systematic methodology study.

The Genome Analysis Toolkit (GATK) was the first Bioinformatics software that applied MapReduce algorithms (McKenna et al., 2010). GATK was a bundle of Java based tools that accepted alignments or assembly results from next

generation sequencing, and provided DNA or RNA rare variation analysis, genotype analysis and related data processing. However, in order to maintain the software to be as widely compatible with all systems as possible, the standard GATK did not take advantage of the Hadoop distributed file system to parallel the program. The software provided several other ways to parallelize the calculation and improve the performance, such as LSF (`http://www.platform.com`) and Sun Grid Engine (`http://gridengine.sunsource.net`). However, those parallelization strategies were limited to specific platforms. Generally speaking, computation efficacy and scalability were not the major consideration of the GATK program.

Crossbow was also among the early attempts to implement the Hadoop and the MapReduce algorithm to analyze large scale genomic data (Langmead et al., 2009). As an effort to broaden the applicability of the software, increase reproducibility and reduce the cost, Crossbow was designed to run on cloud computing (Amazon's EC2 service). Crossbow was a variation analysis pipeline tool that combined short read aligner Bowtie (Langmead & Salzberg, 2012) and variation caller SOAPsnp (Li et al., 2009), with Bowtie being the map phase, SOAPsnp being the reduce phase, and Hadoop's generic sort/shuffle engine in between to order the alignments (resulted from the map phase) according to their genomic locations and feed them to the reduce phase. Using a 320 core cluster system, it was reported that Crossbow was able to analyze the single nucleotide polymorphisms (SNPs) of a 37-fold human genome, which consisted of a mixture of more than three billion single-end and paired-end reads, or 110 Gb of compressed sequence data, within four hours (including file transfer time) with very high accuracy and a cost below $100. This was a huge efficiency improvement compared to other alignment and variation analysis tools on the market at that moment, but the scalability performance of the software was not mentioned.

Recently, BioPig (Nordberg et al., 2013) and SeqPig (Schumacher et al., 2014), which were two very similar but independently developed Hadoop based sequencing data processing platforms, were published. Both BioPig and SeqPig took

advantage of Hadoop's high level data flow language Apache Pig (hence the names) and greatly reduced the programming complexity of building series of mappers and reducers in Hadoop. As a result, those two platforms enabled end users with low programming experience levels to customize their own Hadoop analysis pipeline from provided modules and analyze their sequencing data. In addition, both programs demonstrated horizontal and near-linear scalability and fairly good portability.

Besides genomic analysis, proteomic analysis is another important component of Bioinformatics studies. Typically, proteomics data consist of mass spectrometry results, and the purpose of analyzing proteomics data is to identify the molecular compositions of biological samples, which could contain millions of heterogeneous protein molecules (Cravatt, Simon, & Yates III, 2007; Wright, Noirel, Ow, & Fazeli, 2012). The analysis of proteomics data typically involves spectral deconvolution, peak alignments, quality assurance and database query steps (Zhang, Asara, Adamec, Ouzzani, & Elmagarmid, 2005). Because 1) the composition of protein molecules are much more complicated than the composition of DNA and RNA molecules; 2) the existence of various pre- and post-translational modifications; and 3) the variability the mass spectrometry experiments could potentially bring to the data, the analysis of proteomics data was reported to be more computational intensive than the analysis of genomic data (Wright et al., 2012).

MR-Tandem, which was published in 2012, was the first Hadoop implementation of proteomic analysis tool (Pratt, Howbert, Tasman, & Nilsson, 2012). This program modified the existing MPI code that was used in X!!Tandem (Bjornson et al., 2008). Although MR-tandem ran in Hadoop, it was still based on MPI theory, and thus failed to take advantages of Hadoop's MapReduce algorithm. As an effort to improve MR-Tandem, Hydra was developed (Lewis et al., 2012). Hydra was specifically designed to run on Hadoop and took full advantage of the critical features of the MapReduce algorithm and HDFS architecture. The strategy Hydra conducted to improve efficiency was to proceed from not only the input

queries end, but also the database end. An indexed database with the m/z values of all candidate peptide sequences and all possible modified forms was precomputed via MapReduce and stored in memory to ensure optimal performance. Compared to its serial and MPI counterparts, Hydra showed not only improved efficiency, but also excellent scalability.

2.5 Summary

This chapter provided an overview of relevant literatures for the development and the recent attempts of applying Hadoop in Bioinformatics software. The advantages, concerns and limitations of using Hadoop as a framework in Bioinformatics program design, the basic concepts and application of sequence similarity comparison and digital DNA library subtraction were also presented. By reviewing published works, the researcher pointed out that 1) Hadoop could effectively decrease the programming complexity of parallelizing Bioinformatics programs; 2) a Hadoop version of the sequence similarity comparison tool, the digital DNA subtraction tool, and the sequence de-duplication tool has not been created and is in great demand; and 3) a systematic methodology study is needed.

CHAPTER 3. FRAMEWORK AND METHODOLOGY

The above two chapters introduced the general scope and innovation of the proposed project, and summarized the current achievements and existing gaps reported by prior literatures. This chapter dives into the framework and methodologies to be used in the proposed study.

3.1 Research Framework and Questions

The aim of this project was to improve the efficiency, scalability and usability of a set of Bioinformatics software by 1) automating the data pre-processing, analysis, and the post-processing steps, 2) parallelizing the software by taking advantage of the Hadoop's MapReduce algorithms and 3) building a user friend graphical interface. Because the researcher was interested in studying the computation accuracy, as well as efficiency and scalability of the proposed programs, the research approach used in this project was quantitative. Details about the sampling strategies, data collection and analytics protocols are described in the following sections.

The described quantitative research was designed to test the following hypotheses:

1. The Hadoop MapReduce framework and distributed file system could be utilized to build Bioinformatics applications that automated and improved the following processes: 1) DNA/peptide sequence similarity comparison; 2) digital DNA sequence/library subtraction; and 3) DNA/peptide sequence de-duplication.

2. Compared to other parallelization strategies, and the serial version, the Hadoop implementation of the described programs provided not only the same calculation accuracy, but also significant benefits from scalability and execution speed perspectives.

3. Hadoop could be used as a general tool to parallelize and optimize the performance of Bioinformatics programs.

3.2 Data Source

The following section describes the source of the genomic database, the testing data, as well as the source of the serial programs that the proposed programs were developed from.

3.2.1 Database

In order to perform genome comparison, database (libraries) were needed. In this project, two kinds of libraries were involved:

- Libraries prepared from testing genomes at runtime. For example, in the Digital DNA Library Subtraction tool, when genome A and genome B were compared, genome B was queried against a library prepared from genome A; then genome A was queried against a library prepared from genome B. The programs that were used to prepare libraries from raw sequence data would be introduced in the following sections.

- Known libraries. When testing sequence data was compared to known genomes, non-redundant DNA libraries, which were downloaded from NCBI (`ftp://ftp.ncbi.nlm.nih.gov/blast/db/`), were used.

3.2.2 Testing Data

The comparison of the following genomes were performed in order to test the accuracy, efficiency and the scalability of the new Hadoop programs.

- Small scale.

  Two short sequences files (in the FASTA format) generated from next generation sequencing technique (Mardis, 2008) were used as small scale testing datasets. Each file was 69 Kb in size, contained 2000 short DNA sequences, and each sequence was shorter than 40 bp (base pair) long. There were some low level single nucleotide polymorphism (Ahmadian et al., 2000) (minor variance) between the two files, which we aimed to identify.

- Median scale.

  The cDNA libraries of the Fathead minnow and adult Pimephales promelas (Kane et al., 2008) were downloaded from the NCBI website. The search of NCBI EST (Expressed Sequence Tag) database (`http://www.ncbi.nlm.nih.gov/nucest`) with the keyword "Pimephales promelas" resulted in 258529 cDNA sequences. Within those sequences, 4109 contained the keyword "fry" or "minnow". As a result, the 4109 sequences were downloaded as a FASTA sequence file as the Fathead minnow cDNA library (3.6 Mb) and the rest were downloaded as the adult cDNA library (221.9 Mb).

- Large scale.

  The cDNA libraries of healthy human and human breast cancer patients were downloaded from NCBI website. The search of NCBI EST database with keyword "Homo sapiens" resulted in 8863973 hits, within which 49004 sequences contained key word "breast cancer". As a result, those 49004 sequences were downloaded as a FASTA sequence file as the breast cancer cDNA library (39.0 Mb), and the rest were downloaded as the normal human cDNA library (5.9 Gb).

- Huge scale.

  In order to further test the scalability of the program, extra large scale FASTA sequence files were created artificially. A 17.7 Gb dataset was created by 3x random resampling of the normal human genome data with replacement; while another 3.9 Gb dataset was created by 100x random resampling of the human breast cancer data with replacement.

3.2.3 Serial and MPI Code

The source code of the serial version of BLAST (noted as NCBI BLAST below, version 2.3.0), based on which all three Hadoop programs were designed and implemented, were downloaded from `ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/`. The mpiBLAST (version 1.6.0), with which the efficiency and the scalability of the new programs were compared to, were downloaded from `http://www.mpiblast.org/Downloads/Stable`. Both software were installed under the Linux operation system on both Hathi and Conte, and executed as command line function calls. The source code included nucleotide and peptide library construction tool, nucleotide sequence comparison tool (blastn), peptide sequence comparison tool (blastp), multiple sequence alignment tool (psiBLAST) and so on (Madden, 2013).

3.3 Computation Resource

All Hadoop related computations were performed on Hathi (hathi.rcac.purdue.edu), which is a shared Hadoop cluster operated by Purdue Research Computing, and is a shared resource available to users in Purdue's Community Cluster Program (`https://www.rcac.purdue.edu/compute/hathi/`). Hathi, which went into production in September 2014, consists of 6 Dell compute

nodes with two 8-core Intel E5-2650v2 CPUs, 32 GB of memory, and 48 TB of local storage per node for a total cluster capacity of 288 TB. All nodes has 40 Gigabit Ethernet interconnects with buildin MapReduce framework and Hadoop Distributed File System (HDFS).

All computations not related to Hadoop, for example, the serial and MPI jobs, were performed on Conte (conte.rcac.purdue.edu), which was the largest computer cluster of Purdue's flagship community clusters when this study began. Conte, which went into production in June 2013, consisted of HP computer nodes with two 8-core Intel Xeon-E5 processors (16 cores per node) and 64 GB of memory. Each node is also equipped with two 60-core Xeon Phi co-processors. All nodes has 40 Gbps high speed FDR10 Infiniband connections.

3.4 Procedures and Details

The following procedures were followed when the proposed programs were designed and implemented:

1. Python environment setup and NCBI BLAST tool installation in both the local and the distributed environment.

2. In order to take advantage of Hadoop's MapReduce algorithm, the original serial implementation and algorithms were studied, and necessary mappers and reducers were proposed.

3. Proper pre-processing scripts were created in Python, so that the input FASTA format sequences data was converted to key-value pair format, which was compatible with the MapReduce framework.

4. Mappers and reducers were created and optimized in Python programming language and tested locally with the small scale testing dataset.

5. Proper post-processing script was created so that output results were transfered back from HDFS to local file system, and populated properly into separate FASTA files with appropriate names.

6. Mappers, reducers, pre- and post-processing scripts, as well as the testing datasets were then uploaded to the data nodes of Hathi.

7. The execution time of the Hadoop programs with different testing datasets were documented to represent the efficiency, and the results were compared to those from the serial or MPI programs to demonstrate the accuracy of the Hadoop programs.

8. Several rounds of optimization of mappers and reducers were conducted to further improve the execution efficiency.

9. A user friendly GUI was designed and created by using the TKinter package in Python, in which brief introductions of the three applications were provided, query parameters could be set up by users, and execution process and output results could be observed.

10. Extra-large scale artificial testing datasets were uploaded to Hathi's data nodes and computed by the new Hadoop based Bioinformatics programs using the same number of nodes and the same parallelization parameters. The execution times were documented to demonstrate the scalability of the new programs.

11. MPI programs, whenever available, were utilized to compute the testing datasets (the small scale, median scale, large scale, and huge scale datasets). The results were analyzed and the execution times were plotted and compared to those obtained from the Hadoop programs to illustrate any advances or limitations of the latter in terms of efficiency and scalability.

3.5 Measure of Success

The purpose of the research project was to demonstrate the feasibility of moving the interested Bioinformatics programs to the Hadoop software framework, which should improve both the execution efficiencies and computation scalabilities of the programs. As a result, the successfulness of the project fell into three categories.

The first level of successfulness was achieved if the researcher was able to move all three programs of interest to the Hadoop software framework, while maintaining the integrities of the results as obtained in the serial implementations. In other words, a running version of the Hadoop programs that provided accurate results was a sign of the first level success.

The second level of success was the improvement of execution efficiency. As discussed in the literature review, the MapReduce algorithm is normally not as efficient as MPI parallelizations, because MPI strategies is typically optimized to customize every single situation (Gabriel et al., 2004), while Hadoop is a general purpose parallelization strategy (White, 2012). However, for a successful Hadoop program, a significant speedup with at least 50% less execution time should be observed when compared to similar serial implementation. In addition, the efficiency loss between a Hadoop version and its MPI counterpart should be less than 30%.

The third level of success measured the scalability. As mentioned before, linear scalability is one of the most prominent advantages of Hadoop programs. In this project, after datasets of different sizes were tested and analyzed, the execution times were plotted against the sizes of the testing datasets. A linear or near linear relationship between the two variables was expected. In addition, it was also expected that the scalabilities of Hadoop programs were much better than the MPI programs. In other words, the linearity of the execution time plot of a Hadoop program should be significantly better than that of the corresponding MPI program. A statistical test would be conducted to compare the linearities if the results were close.

3.6 Data Collection and Analysis

The data collection and analysis processes has been partially discussed in Section 3.2.2 and Section 3.4. Specifically, the following data analysis steps were conducted to establish the accuracy, efficiency and scalability performance of the new Hadoop programs.

3.6.1 Accuracy

The Fathead minnow and adult dataset were analyzed by using both the original serial implementation and the new Hadoop version of the interested programs. Specifically, the DNA sequences that can be found in both the minnow and adult fathead fish would be identified by both the serial NCBI BLAST and the Hadoop sequence comparison tool; the Hadoop Digital DNA Sequence/Library Subtraction tool was used to extract all unique DNA sequences that were expressed in minnow but not adult fathead fish, and all sequences that were expressed in adult but not minnow fathead fish; finally, Hadoop sequence de-duplication tool would be executed to combine the two sets of sequences and remove the similar sequences based on user defined similarity requirements. The two sets of results (one from NCBI BLAST, one from the Hadoop programs) would then be compared. Ideally, identical results were to be observed. If there existed any differences, they would be analyzed and traced back to source code.

3.6.2 Efficiency

All collected datasets, including the small scale next generation sequencing data, the median scale minnow and adult fathead data, the large scale human and breast cancer data, and the huge scale artificial data, were used to test the efficiency of the new programs. When the pre- and post-processing scripts were created, the start and end time of the program were printed to the output file intensionally, so that execution time could be obtained easily. Execution time was then analyzed as

the variable that represents efficiencies of the programs. Similarly, the serial version, the MPI version (where available) and the Hadoop version of the same programs were executed to compute results from the above datasets, and the execution times were documented and compared.

3.6.3 Scalability

Similar to the measurement of efficiency, all collected datasets were used to measure the scalability performance of the new Hadoop programs (Schumacher et al., 2014). After all execution times were measured and documented, a plot of execution times against the sizes of the input datasets was created for each version of each individual program (in other words, there would be three plots for Hadoop programs, three plots for serial programs and one for MPI programs). For each of the three programs, the slopes and linearities of the plots from different program versions (serial vs MPI vs Hadoop) were compared and statistical tests, such as paired T-tests or multiply Tukey tests (Lowry, 2014) would be conducted wherever necessary, to establish the significant differences between the results.

3.7 Summary

In this chapter, the research framework of the proposed study, as well as the research questions being answered were discussed. An overview of sampling strategies, proposed procedures to conduct the project, measurements of success and data analysis protocols were also provided.

CHAPTER 4. RESULT

In this chapter, the outcomes of the proposed project are described in details. Specifically, the design and implementation of the mappers and the reducers, the graphical user interface (GUI) of the programs, and the measurements of success are shown.

4.1 The Design of Mappers and Reducers

As discussed in Chapter 3, program optimizations were conducted after the initial design of the MapReduce Algorithm. Shown in this chapter are the algorithms that were implemented in the final version of the three programs. The described algorithm was a balance between available computer resources and the performance of the programs. Generally speaking, the three new Hadoop Bioinformatics programs (the DNA/peptide Sequence Similarity Comparison tool, the Digital DNA Library Subtraction tool, and the Sequence De-duplication tool) shared the same pre-processing and library comparison steps, but in the sequence extraction and post-processing stages, different strategies were used to ensure accurate calculation.

4.1.1 The pre-processing of input data

Two formats of input data were accepted by the new Hadoop programs, namely FASTA sequence files and FASTQ sequence files. In those two types of files, each sequence is represented by one line of unique identifier, one or more lines of sequence strings, and in the case of FASTQ file, one line of sequencing quality

scores. However, the type of input data Hadoop framework expected is in the format of {key, vale} pairs. As a result, four steps of pre-processing were conducted.

4.1.1.1. Combining Multi-Line Sequences and Removing Quality Scores.

When the DNA/peptide sequences are long, they sometimes are written across multiple lines in FASTA or FASTQ files. This could generate serious issues in a Hadoop job, because in the Hadoop framework, each line is treated as an individual data point. As a result, all input files went through a pre-processing step, which converted all individual sequences into a single line format (see example below). In the case of a FASTQ file, the last line of a sequence, which contained the quality scores, was simply removed.

4.1.1.2. The Creation of Key-Value Pairs from Input Sequence Files.

As described above, after combining the multi-line sequences, each sequence still took at least two lines, because the unique ID of the sequence and the sequence string itself were written in two separate lines in FASTA or FASTQ files. As a result, the second step of pre-processing converted the unique ID of each sequence into the key, and the sequence string into the value, and a tab was used to delimit the two fields, all of which were written in a single line.

4.1.1.3. Labeling the Search Libraries.

All three new Bioinformatic programs compared the similarities/differences between two input sequence files. As a result, for each file, the other file was considered as the search library. Another issue with using FASTA/FASTQ files was

that the sequence ID did not reflect which file the sequence came from. However, in Hadoop, all input are taken directly from standard input and the mapper/reducer do not have access to the file name. Therefore, a simplified representation of the input sequences was needed to include all necessary information in standard input. The strategy the researcher took was to include the name of the library file in the key field of each sequence, and combine the {key, value} pairs from the two input files into one file, which was then pass into standard input.

4.1.1.4. Constructing the Sequence Libraries.

Besides input sequences manipulations, NCBI BLAST toolkit command *makeblastdb* was also executed in the pre-processing step. With each sequence file as a function input, this command built the corresponding libraries and index files, which expedited the subsequent query.

4.1.1.5. Example and Other Considerations.

Below is an example that illustrates the type of transformation the first three pre-processing steps did. If the two input sequence files were called seq1.fasta and seq2.fasta, and in the original FASTA file, one of the sequences from seq1.fasta read as:

```
>BG3:2_9M1:191:572/1
AGAATTTCAAGAATTAATGGCCGC
ATGGGAATATC
```

After the three steps of preprocessing, the {key, value} pair format of this sequence with library label was written as:

```
seq2>BG3:2_9M1:191:572/1    AGAATTTCAAGAATTAATGGCCGCATGGGAATATC
```

In the former multi-line representation, the first line was the unique ID of the sequence, which started with a symbol ">". The second and the third lines were the actually DNA sequence. In the latter key-value pair representation, first of all, the second and third lines were combined, so that the DNA sequence was written in a single line; secondly, the ID and the sequence were combined, with a tab in between; finally, "seq2" was added to the beginning of the line, in the front of ">", to specify that in the similarity comparison, seq2.fasta was used as the library file, or in other words, this sequence came from seq1.fasta.

Because the existence of independencies among different lines, the above pre-processing steps had to be executed in serial mode. However, since the time complexity of the algorithm was linear, the execution of the pre-processing code should not be considered as the bottle neck of the whole program.

4.1.2 Mappers and Reducers

The final algorithms that the researcher implemented into the proposed programs consisted of three MapReduce steps. Various processing strategies that had been considered during the optimization stage are discussed in more details in the next chapter. In the following discussion, "\t" was used as a symbol to represent a tab delimiter.

4.1.2.1. Mapper 1

The input key-value pair of the first mapper has been discussed in the pre-processing step. The format is:

```
library>ID\tsequence
```

where the string before tab (library>ID) is the key, and the string after the tab (sequence) is the value. Mapper1 took the input key-value pairs, and the returned key-value pairs were in the following format:

```
library\tID|||sequence
```

where the new key was the name of the library, and the value combines the ID and the sequence, with "|||" as a delimiter. Taking the example above, now the sequence became:

```
seq2    >BG3:2_9M1:191:572/1|||AGAATTTCAAGAATTAATGGCCGCATGGGAATATC
```

The transformation in Mapper 1 took linear time.

4.1.2.2. Reducer 1

The first reducer split the input files into chunks of key-value pairs that contained a specific number of sequences. For example, assuming that seq1.fasta contained 1500 sequences, and seq2.fasta contained 1750 sequences, after the execution of the first mapper, there were two valid keys: seq1, which had 1500 different values; and seq2, which had 1750 values. When key-value pairs were fed into the first reducer, new keys were assigned based on a known (user defined) chuck size. In the above example, if chuck size was defined to be 500, then the following keys would be produced:

- *seq*1__0, *seq*1__1, *seq*1__2, *seq*1__3
- *seq*2__0, *seq*2__1, *seq*2__2, *seq*2__3, *seq*2__4

All keys above contained 500 values, except *seq*2__4, which would contain 250 values. For each sequence, the value kept the same, while the ID was edited to specify which chuck it was categorized to. In addition, the values of different

sequences with the same key were concatenated together and delimited by "|??|". The reason to split input key-value pairs into chunks and how to choose chunk size are discussed in Section 5.1.3. Figure 4.1 also illustrates the notion of chunks in the current data analysis pipeline.

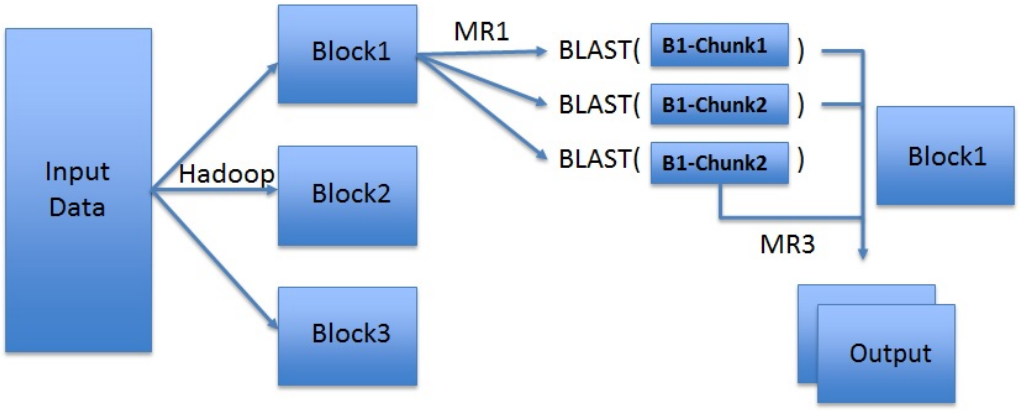

Figure 4.1. A illustration of the MapReduce Algorithm.

B1: Block1;
MR1: Mapper 1 and Reducer 1;
MR3: Mapper 3 and Reducer 3.

4.1.2.3. Mapper 2

Now that we had both the sequence libraries prepared (in the pre-processing step), and the input key-value pairs properly formatted and split into chunks (resulted from mapper 1 and reducer 1), the actual sequence comparison was ready to take place.

The input key-value pairs of mapper 2 came from the output of reducer 1. All input sequences were assigned to specific chunks based on the input key, and

each chuck was treated as a query file and compared to a certain library based on the first part of the key that had the library name encoded.

For each comparison, the library files were loaded to memory, then, a polynomial process with n^2 time complexity took place, where every sequence from the input chunk of key-value pairs was compared to each sequence from the library file. Based on user defined similarity level, if a input key-value pair had a match in the library (that was beyond the similarity score), the unique ID of the sequence from the library file was printed to the standard output. This process was repeated for all input sequences.

As mentioned above, in this step, a user defined similarity level was needed by the program. The program provided users with two options, percentage similarity between the two sequences and E-value, which measures the confidence level of the match. Percentage similarity compares the bitwise similarities between a query sequence and a library sequence. This method shows the following two drawbacks.

- The lengths of the two sequences have significant impact on the similarity results.

  This is not always true when two DNA or protein sequences are compared. For example, if one sequence were a truncated product of the other, their length, and thus percentage similarity would differ a lot, but by all means they should be considered "similar".

- All bitwise differences are penalized the same way.

  As discussed in the literature review (Section 2.3), a lot of synonymous point mutations exist in biological molecules like DNA and protein. As a result, the biological effect of one single bitwise difference between two sequences could be anything between completely benign and fatal. In other words, treating all the bitwise differences in the same way results in the lose of important biological information.

Because of the above limitations, another similarity evaluation mechanism, E-value is used more frequently by Bioinformaticians. E-value takes into considerations the statistical distribution of the differences between sequences, and treated purine to purine and pyrimidine to pyrimidine mutations differently from purine to pyrimidine or pyrimidine to purine mutations (Schäffer et al., 2001). On the other hand, however, calculating E-values is more time and memory consuming than comparing the percentage similarities, as the underlying distribution needs to be constructed in each comparison.

In this step, because the computation of each chunk of sequence data required the loading of one copy of the libraries, and that the data were replicated across multiple computer nodes, the memory usage was high when libraries were large, chunk size was large, or too many mappers were deployed at the same time. As a result, the default chunk size of the program was set to 2000, and the maximum number of mappers was restricted to 20.

In addition, mapper 2 was a mapper-only step without any following reducer steps. The output key-value pairs of mapper 2 were in the following format:

```
file_name\tID
```

In our running example, if the sequence AGAATTTCAAGAATTAATGGCCGCATGGGAATATC, which came from seq1.fasta, was found to be a match by a query from seq2.fasta based on the given similarity level, then the output key-value pair for this sequence after mapper 2 step would be:

```
seq1    BG3:2_9M1:191:572/1
```

All key-value pairs of matched sequences from the above library search were written to files on HDFS, which were then merged and downloaded to local file system, converted to a Python dictionary, with file names as dictionary keys, and another dictionary of ID names as values. The dictionary was exported as a pickle

file (Python readable file that stored Python objects), so that it could be loaded to the next map-reduce step easily and efficiently.

4.1.2.4. Mapper 3

A list of IDs of similar sequences was generated from mapper 2. In mapper 3 and reducer 3, the three new Bioinformatics programs, digital DNA library subtraction, sequence de-duplication and sequence similarity comparison extracted different information from the original sequence files based on this list, and associated the extracted information with proper output file names. As a result, three mappers were created in this step, one for each program.

Generally speaking, the input key-value pairs of mapper 3 were the same as the input of mapper 1, which contained the library tag, unique ID and sequence information from both input files, and were in the following format:

```
library>ID\tsequence
```

The list of identified IDs from mapper 2, which was stored in a pickle file, was used as a "look up table", so that desired sequences could be extracted.

The output key-value pairs of this step were in the following format:

```
file_name|||status\t>ID\tsequence
```

where file_name denoted the prefix of the input file, and status was either "shared" or "unique", based on whether or not the ID of the sequence appeared in the output of mapper 2. The following paragraphs explained the detailed algorithms of the three mappers.

- Sequence Similarity Comparison

  This program aimed at extracting the similar sequences between the two input files. As a result, the ID, sequence, and file name information of the sequences

that were identified by mapper 2 were displayed in the standard output. For example, if the names of the two input files were seq1.fasta and seq2.fasta, then the two keys that were generated from this step were “seq1|||shared” and “seq2|||shared”.

- Diginal DNA Library Extraction

  This program aimed at extracting sequences that could be found in one input file but not the other. As a result, the ID, sequence, and file name information of the sequences that were not identified by mapper 2 were printed to the standard output. For example, if the names of the two input files were seq1.fasta and seq2.fasta, then the two keys that were generated from this step were “seq1|||unique” and “seq2|||unique”.

- Sequence De-duplication

  This program aimed at combining the two input files while keeping only one copy of the similar sequences, based on the user defined similarity level. As a result, the ID, sequence, and file name information of all the sequences, both the ones identified and the ones not identified by mapper 2 were printed to the standard output. For example, if the names of the two input files were seq1.fasta and seq2.fasta, then there would exist four possible output keys after this step:

  - “seq1|||unique”
  - “seq2|||unique”
  - “seq1|||shared”
  - “seq2|||shared”

4.1.2.5. Reducer 3

In this step, the Sequence Similarity Comparison tool and the Diginal DNA Library Extraction tool shared the same reducer script, while another script was created to accomplish the sequence de-duplication task. Generally speaking, the input key-value pairs were resulted from mapper 3, and the format of the output keys varied between the two scripts.

- Sequence Similarity Comparison and Diginal DNA Library Extraction.

  In this reducer script, the ID and sequence information of the sequences that shared the same input key were concatenated together. The output keys were the same as the names of the output files, and the values of each key included all the contents that needed to be written to the corresponding output file. For example, if a digital library extraction job between seq1.fasta and seq2.fasta was executed, and 500 unique sequences were identified from seq1.fasta, and 600 unique sequences were identified from seq2.fasta, then the two output keys after reducer 3 would be:

  ```
  "seq1\_unique.fasta"
  ```

  and

  ```
  "seq2\_unique.fasta"
  ```

  The value of the former key would be a long string that included the ID and sequences information of all the 500 sequences with delimiters, and the value of the latter key would be a long string that included the information of the 600 sequences with delimiters. In other words, only 2 key-value pairs would be created by reducer 3 (in both the case of Sequence Similarity Comparison tool and the Digital DNA Library Extraction tool).

- Sequence De-duplication

  Because only one output file that included the combined sequences from the two input files was required, the third reducer step of the Sequence De-duplication tool could be simplified to a large extent. In this step, there were four possible keys from the input key-value pairs, all of which had the following format:

  ```
  file_name|||status
  ```

  In the above format, the "status" of the keys could be either "shared" or "unique". The ID and sequence information of all the key-value pairs whose key contained status "unique" was redirected to the standard output. For the key-value pairs whose key contained status "shared", only the ID and sequence information from the first key was printed out, and the sequences from the second key were simply removed, as they represented the "duplicated" information. Then, when the output files were merged and downloaded from HDFS to local file system, they were renamed to be dedup_file1_file2.fasta. In other words, the Sequence De-duplication tool did not require any additional post-processing steps.

4.1.3 The Post-Process of Output Data

As discussed above, the Sequence De-duplication program did not require any post-processing. On the other hand, it was necessary for the Sequence Similarity Comparison tool and the Diginal DNA Library Extraction tool, because multiple output files were expected.

For the latter two programs, the output from reducer 3 contained two key-value pairs. Each key corresponded to a output file name, while the value was the contents of that file. As a result, the post-processing script simply took the

result from reducer 3, skimmed through the file, open an output file with a proper name when it reached a key, and wrote all the following values to that file.

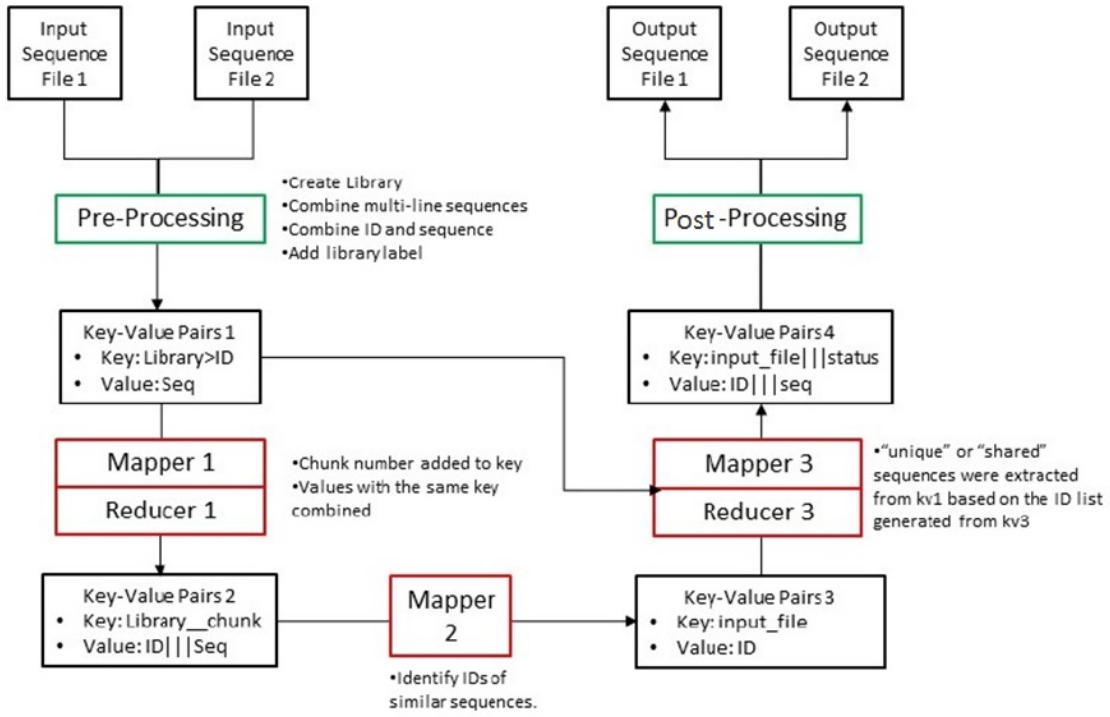

Figure 4.2. The data processing pipeline.

Similar to the pre-processing, the post-processing step was accomplished locally as a serial code, but because it only required linear execution time and memory, the execution should be relatively fast.

Figure 4.2 shows the overall pipeline where the two input sequence files (FASTA format or FASTQ format) went through the pre-processing, mapper 1 and reducer 1, mapper 2, mapper 3 and reducer 3, and the post-processing (optional) to give the output sequence file/files. In this figure, the input/output/intermediate files are shown in text boxes with black frames; MapReduce steps are shown in text boxes with red frames; and local serial processing steps are shown in text boxes with green frames. The format of the key-value pairs is specified within each black text

box. The transformation or selection operation that was conducted in each step is denoted besides each red/green text box.

4.2 The Graphical User Interface

The above sections described the overall logic of the MapReduce algorithms that compared DNA/peptide sequences behind the three proposed Bioinformatics programs. However, in order to execute pipeline, which contained two pre-processing scripts, three mappers, two reducers, and one post-processing script, one had to have a relatively deep understanding of the algorithm, the input and output files, as well as the scientific meaning of all the input parameters, which could be a somewhat challenging task for normal users. As a result, a Graphical User Interface (GUI) was designed and implemented as the front end interface of the three programs to help the users build and manage their own pipelines and choose the most appropriate parameters. The user friendly features were designed based on the research's recent publication (Wang & Springer, 2015) and are discussed in Section 5.2.

4.2.1 The Front Page

Figure 4.3 shows the front page of the GUI. This page was divided into two group boxes. The first group box contained a short paragraph of introductory information, which illustrated the basic functionalities of this toolkit. If users were interested in one specific program, they could click into the button with the name of the program, and find detailed information.

4.2.2 The Introductory Pages

Figure 4.4, Figure 4.5 and Figure 4.6 show the introductory pages of the three Bioinformatics programs, the Sequence Similarity Comparison tool, the Digital DNA Library Subtraction tool and the Sequence De-duplication tool. Each

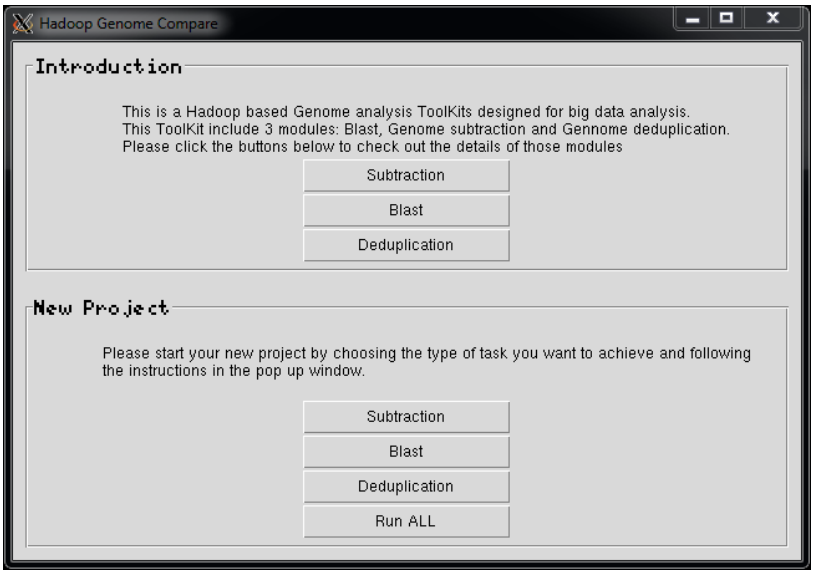

Figure 4.3. The front page of the GUI.

of these pages included information such as the required format of the input files, the general transformation the program executed, the similarity levels the users could choose from, the contents of the output files, as well as a colorful Venn diagram that illustrated the kind of comparison of the program. In the Venn diagrams, each color represented one output file.

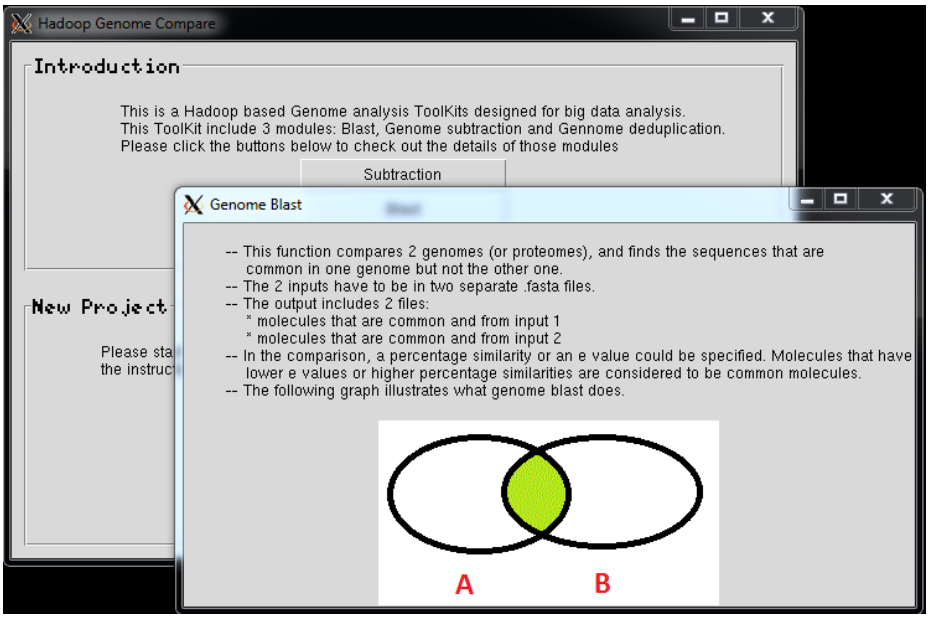

Figure 4.4. The introductory page of sequence similarity comparison.

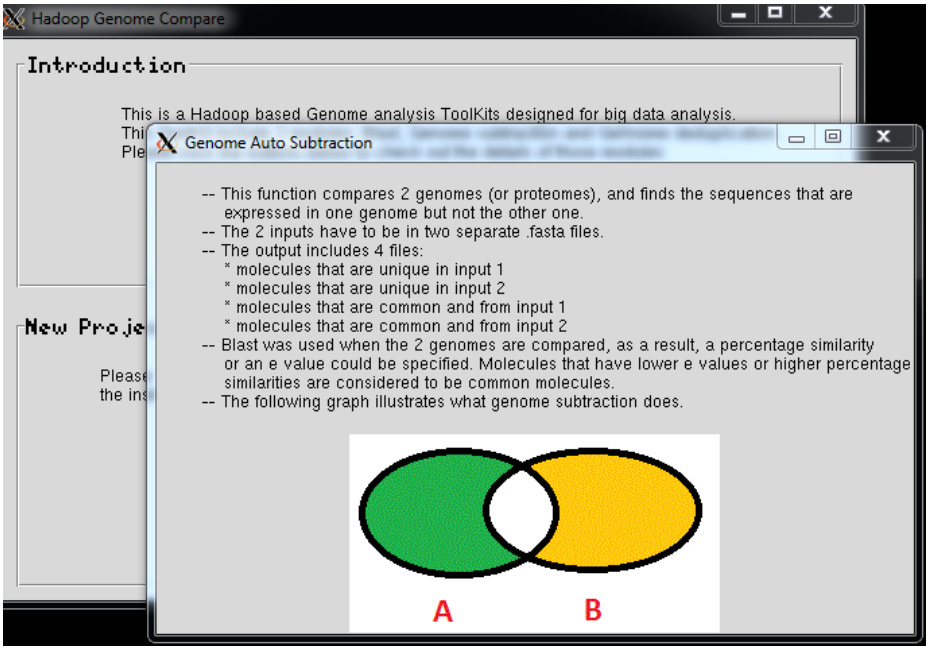

Figure 4.5. The introductory page of digital library subtraction.

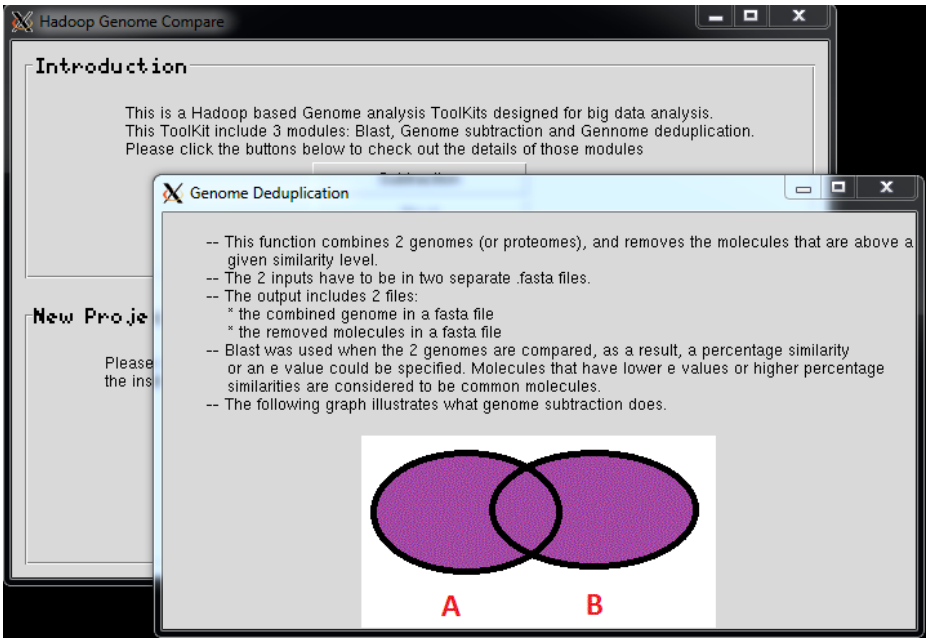

Figure 4.6. The introductory page of sequence de-duplication.

4.2.3 The Execution Pages

Figure 4.7, Figure 4.8 and Figure 4.9 show empty execution pages of the three programs. Figure 4.10 shows the empty execution page that generated comparison files from all three programs. Each of those execution pages was split into three group boxes.

- Input Files.

  Users need to choose the format of the input file from one of the two radio buttons, FASTA or FASTQ. In addition, there are two “Browse...” buttons where one can browse the local system and upload input sequence files. The users are also given the option to specify the chunk size and the name of the output directory. The last two fields, chunk size and output file name are optional. In other words, default values are provided, so that even if the user do not define any values, the program will still run. Figure 4.11 shows a sample page where users can choose files and upload to server.

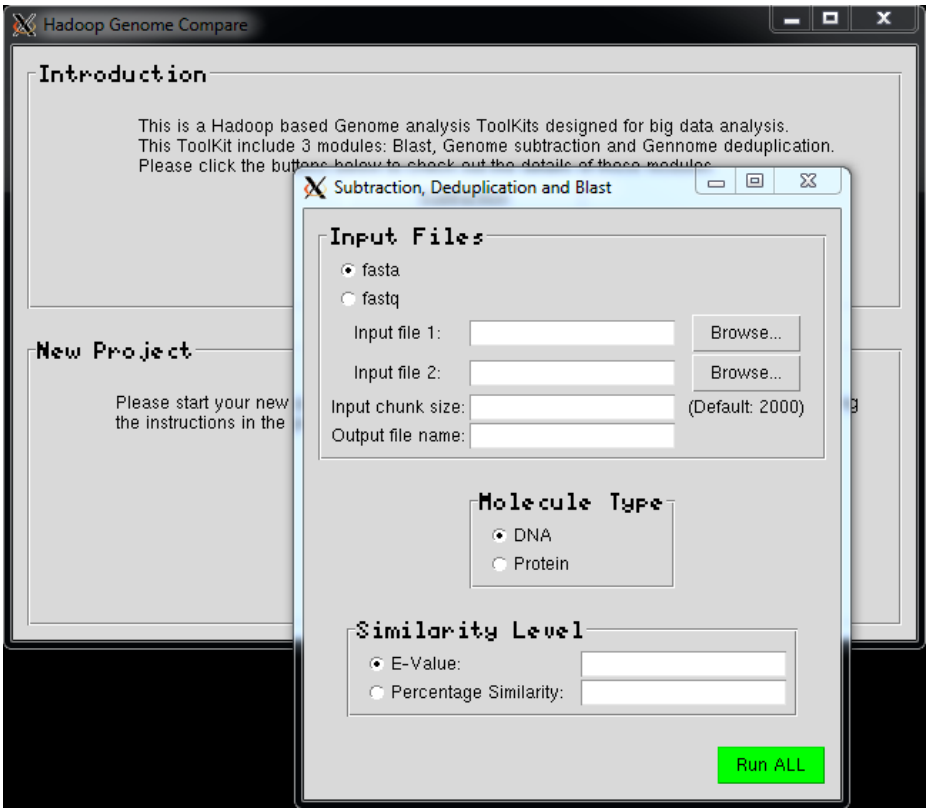

Figure 4.7. The execution page of generating all comparison files.

- Molecule Type

  Two options, DNA and Protein are provided as radio buttons, and the default option is DNA.

- Similarity Level

  As described above, two similarity options, E-Value and Percentage Similarity are provided. The user needs to first choose the similarity type from the radio buttons, and then give a cutoff value to the following textbox. Default cutoff values are also provided, so that the program will not crash even if no cutoff values were provided by the user.

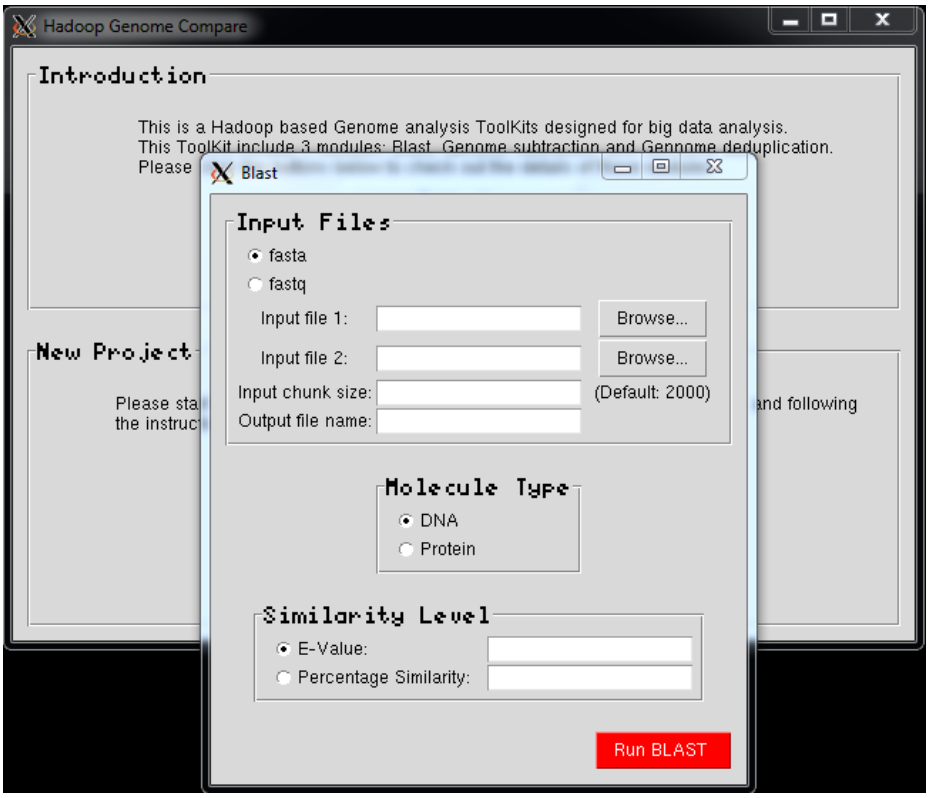

Figure 4.8. The execution page of generating the sequence similarity comparison files.

4.2.4 The Status Pages

After the "Run" button from the execution page is clicked, a status page that dynamically updates the execution status appear. Figure 4.12 shows an example of the status page when a sequence similarity comparison job is running. Figure 4.13 shows an example of the status page when a sequence similarity comparison job completes with running errors.

Figure 4.14 shows an example of the status page when a job that executes all three programs completes, and new files are successfully generated.

A status page provides the following information.

- The start and end time of the job.

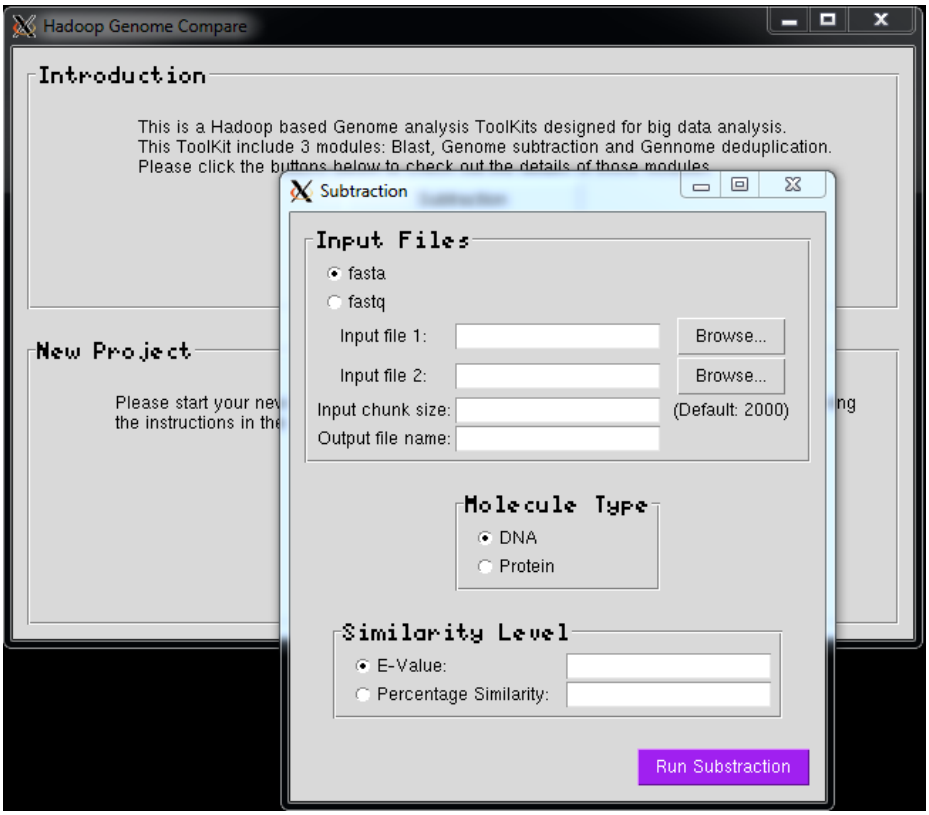

Figure 4.9. The execution page of generating digital library subtraction files.

- The current status of the job. In other words, which step is running and which steps have been completed.
- The name and directory of output files.
- The final status of the job, whether it succeeded or failed.

In addition, the GUI was designed to include some user-friendly features. The details of those features are discussed in Section 5.2.

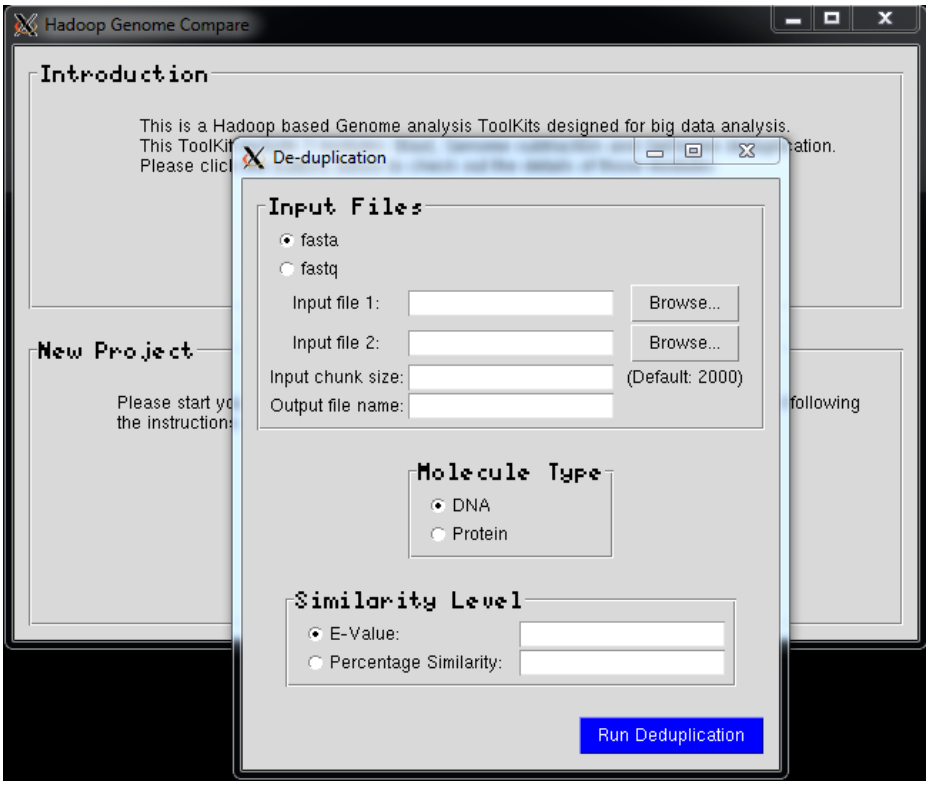

Figure 4.10. The execution page of generating the sequence de-duplication files.

4.3 The Measure of Accuracy, Efficiency and Scalability

As it has been discussed in Section 3.5 Measure of Success, there were three levels of success in this project. The first level is accuracy, which requires that all three programs are moved to the Hadoop software framework while the integrities of the calculation results are maintained. The second level is efficiency, which means by introducing the MapReduce algorithm, the execution time improved. The third level is scalability, which requires that the accuracy and efficiency gain scales well with the size of input data.

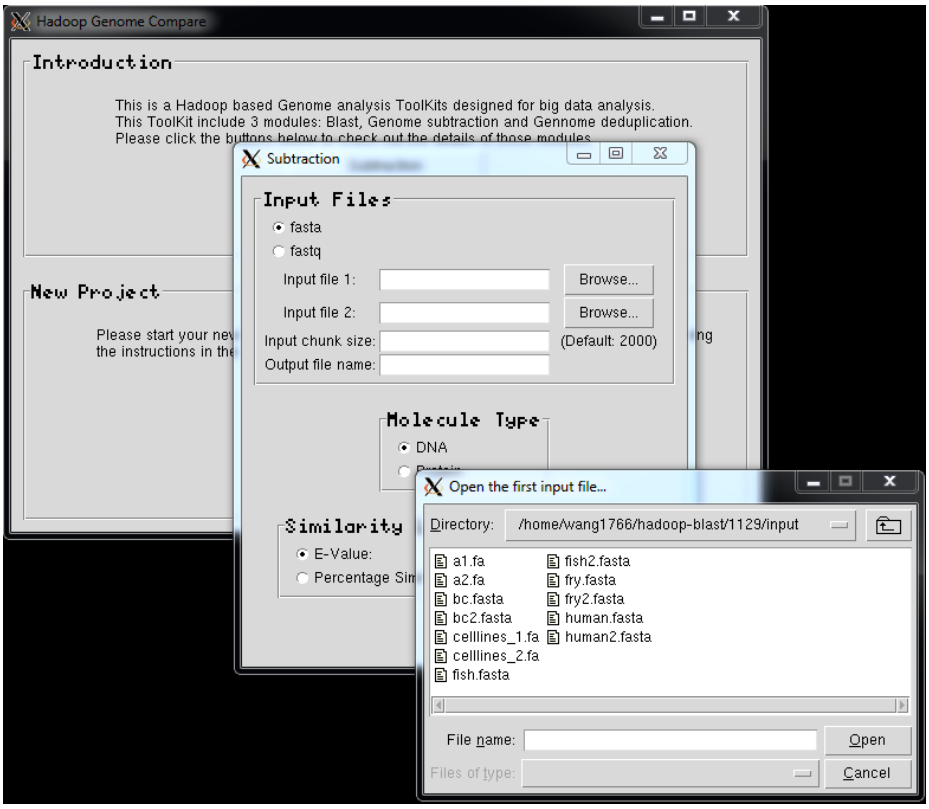

Figure 4.11. The file uploading page.

4.3.1 Accuracy

4.3.1.1. Testing Data

In order to measure the accuracy of the new Hadoop programs, output results from the Hadoop version of the Sequence Similarity Comparison tool were compared to those from the serial version of NCBI BLAST. The small scale next generation sequencing data and median scale Fathead minnow and adult cDNA data mention in section 3.2.2 were used in this comparison.

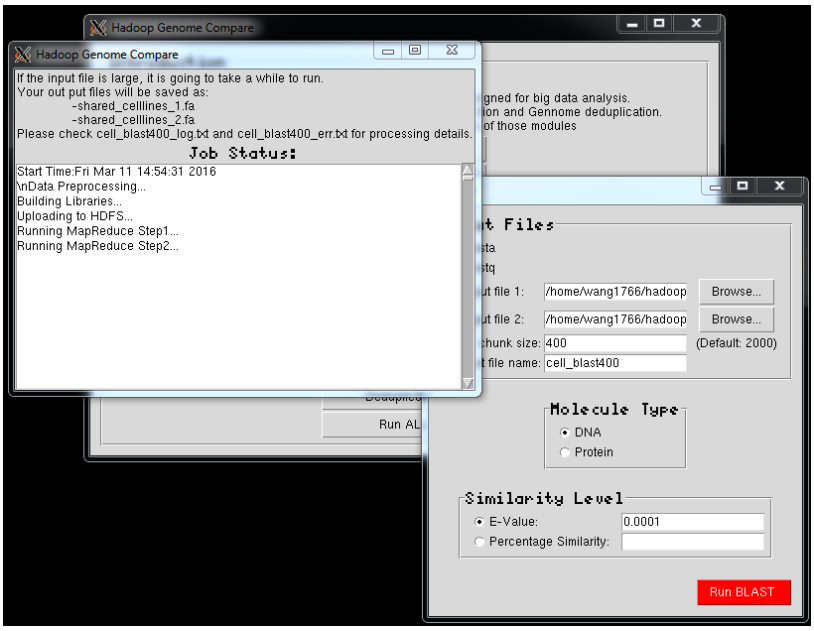

Figure 4.12. The status page of a running job.

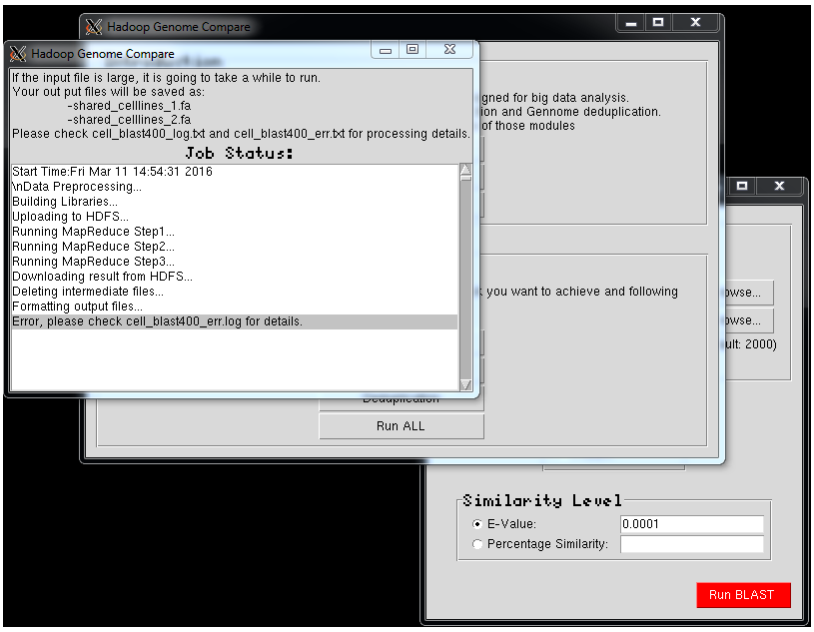

Figure 4.13. The status page of a failure job.

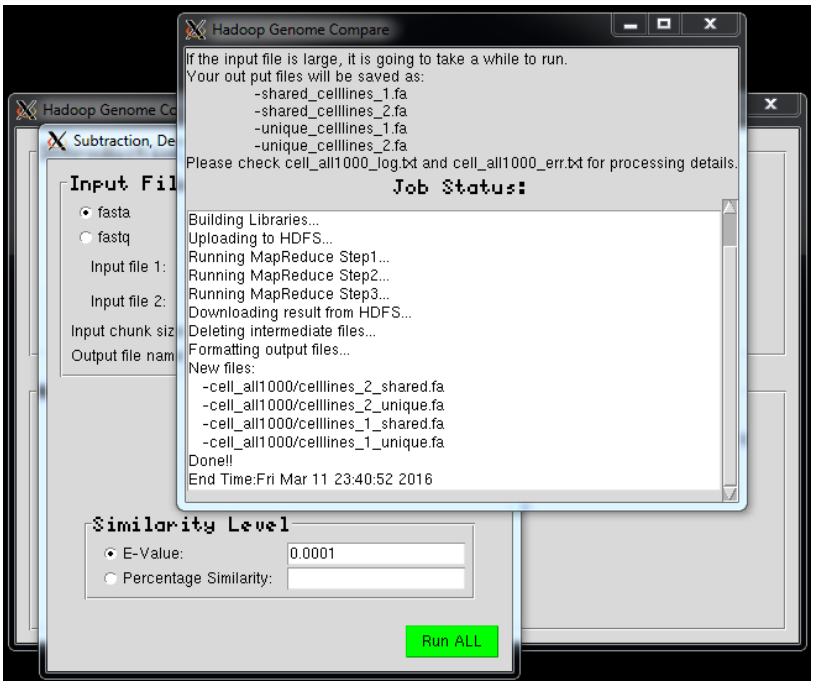

Figure 4.14. The status page of a successful job.

4.3.1.2. Testing program

The reasons that only the Hadoop Sequence Similarity Comparison tool was chosen to accomplish the accuracy measurement included:

- The three programs only differ on mapper 3, reducer 3, and the post-processing steps. The main comparison step was carried out in mapper 2, which the three programs shared.
- Both the serial version of the Sequence Similarity Comparison tool, called NCBI BLAST (S. F. Altschul et al., 1990), and a previous parallelized version, called mpiBLAST (Darling et al., 2003) were widely used and readily available.

It should be noted that NCBI BLAST is an open source alignment and search tool, for the purpose of identifying the similar sequences between two DNA

or protein libraries, additional pre-processing steps, such as the construction of libraries, and the combining of multi-line sequences; and several post-processing steps, such as extracting the sequences based on the identified IDs, reformatting IDs and sequences in the output files needed to be hand coded.

4.3.1.3. Results

The comparison of the output results took place at two levels.

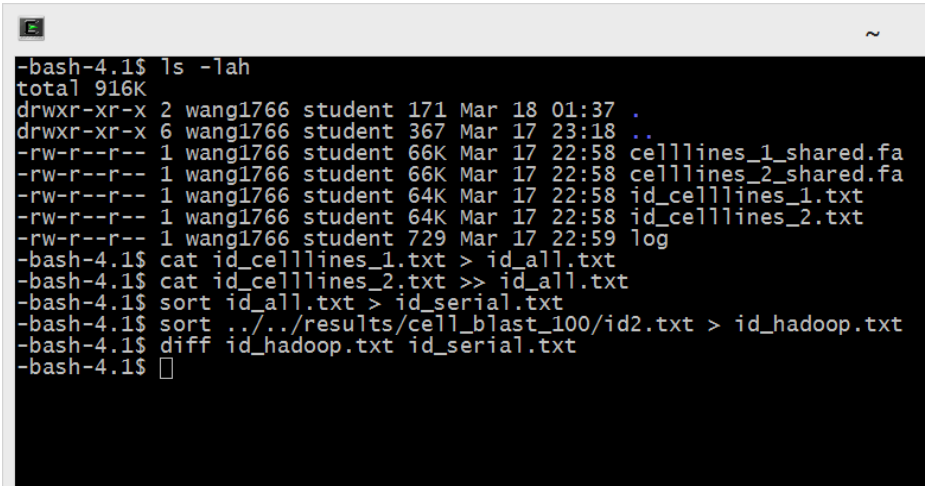

Figure 4.15. The comparison of unique ID list generated from NCBI BLAST and Hadoop sequence similarity comparison tool.

(Data set: small scale next generation sequencing data)

- First of all, the unique ID list that were identified by the serial BLAST tool and the mapper 2 of the Hadoop Sequence Similarity Comparison tool were compared. The results are shown in Figure 4.15 and Figure 4.16. In both cases, the two input files were queried against each other using serial NCBI BLAST, which generated two lists of unique IDs. The two lists were combined together and sorted, which was compared to the sorted result generated from

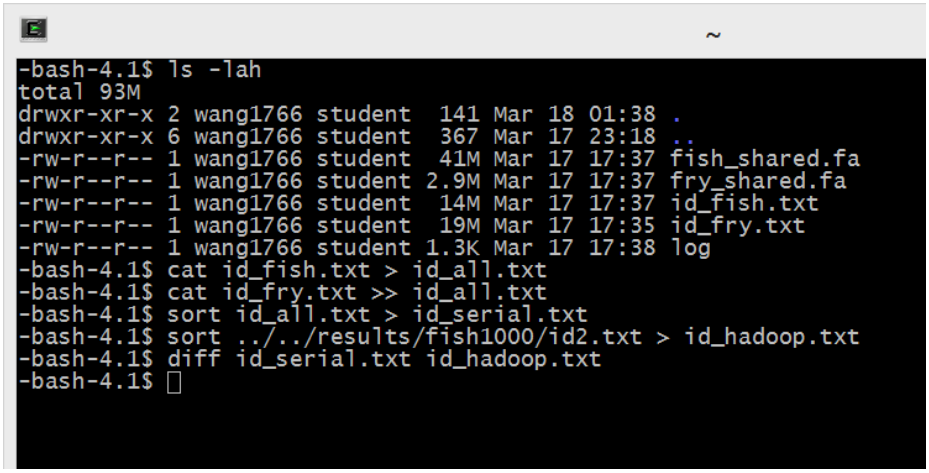

Figure 4.16. The comparison of unique ID list generated from NCBI BLAST and Hadoop sequence similarity comparison tool.

(Data set: median scale Fathead minnow and adult cDNA data)

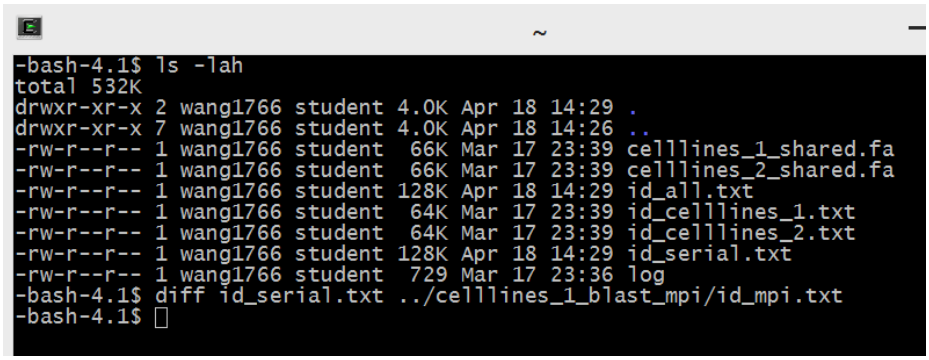

Figure 4.17. The comparison of unique ID list generated from NCBI BLAST and mpiBLAST.

(Data set: small scale next generation sequencing data)

the mapper 2 step of the Hadoop Sequence Similarity Comparison tool. It should be noted that only the value part of the output key-value pairs of mapper 2 were used in the comparison. In other word, the library tags and the '>' delimiters were removed prior to sorting and comparison. It can be easily observed from Figure 4.15 and Figure 4.16 that in both the case of small

scale dataset and median scale dataset, after sorting, the two ID lists were completely the same, since the "diff" function call from Linux shell did not return any values. Furthermore, in order to assure that the query parameters of the mpiBLAST and the NCBI BLAST were the same, and thus not impacting the comparison of efficiency and scalability described below, the output results from mpiBLAST was also compared. As shown in Figure 4.17, the outcomes were the same.

- After post-processing, the output sequence files from the serial version and the Hadoop version of the application were compared again. After sorting, the "diff" function call from Linux shell was used to compare the two output FASTA files, which were completely the same, since the function call had no return values.

In conclusion, the output results from the Hadoop Sequence Comparison tool and the NCBI BLAST tool were the identical. In other words, the calculation accuracy, which was considered as the first level of successfulness in this project, was achieved.

4.3.2 Efficiency

4.3.2.1. Testing Program

In order to measure efficiency, the second level of successfulness of the new Hadoop programs, the running time from the Hadoop version of Sequence Similarity Comparison tool were documented and compared to two other programs: the serial version NCBI BLAST, and an earlier parallelized implementation, mpiBLAST (Darling et al., 2003). Taking advantages of MPI (Message Passing Interface), a widely used parallelization strategy that utilizes distributed computational

resources, the latest version, mpiBLAST-1.6.0 was reported to improve the performance of NCBI BLAST by several orders of magnitude (Lin, Ma, Feng, & Samatova, 2011).

The reason that only the Hadoop Sequence Similarity Comparison tool was used in the comparison of efficiency were described above: the three new Bioinformatics programs only differ on the last stage, and that both the serial and parallelized version of the program were open source and readily available.

4.3.2.2. Testing Data

In this test, all four data sets described in Section 3.2.2, including the small scale next generation sequencing data, the median scale Fathead minnow and adult cDNA data, the large scale human normal and breast cancer data, and the ultra-large scale artificial DNA data, were used in the comparison.

4.3.2.3. Data Collection

As described in Section 4.2.4, the GUI of Hadoop sequence similarity comparison tool displayed the start and finish time of each comparison job. In addition, this information was also documented in the log file of each job. For ncbi- and mpiBLAST, shell script

```
echo "$(date +"%T")"
```

was used at the beginning and end of the wrapper script to enable the calculation of the total execution time, which included the pre-processing of the input files, the comparison of sequences, and the post-processing of the output results. To guarantee fair comparisons, a maximum of 20 mappers in the Hadoop program, and 20 computation threads in the mpiBLAST were enforced.

4.3.2.4. Results

The execution times of different datasets using the three versions of the program are listed in Table 4.1 below. The two columns on the left denotes the dataset and the size, and the three columns on the right denotes the total run time from different programs.

Table 4.1
Execution Times of Sequence Similarity Comparison Tasks with Different Programs.

Input		***Program Run Time***		
Dataset	Size	NCBI BLAST	mpiBLAST	Hadoop Version
Small	69 Kb + 69 Kb	00:00:02	00:00:17	00:03:45
Median	221.9 Mb + 3.6 Mb	00:07:38	00:03:59	00:07:25
Large	5.9 Gb + 39.0 Mb	03:38:22	01:01:42	01:29:51
Huge	17.7 Gb + 3.9 Gb	48:34:48	08:02:15	05:50:09

The results clearly showed that for the small scale next generation sequencing data with 2000 sequences in each input file, NCBI BLAST was the most efficient program. mpiBLAST and the new Hadoop Sequence Comparison tool were not as fast, probably because of latency time. It can be observed that the latency time of the Hadoop program was quite significant, probably because all input, output and some of the intermediate files needed to be transfered between HDFS and the local file system. mpiBLAST also had latency time, but it appeared to be much shorter than the latency time of the Hadoop program.

For the median scale Fathead minnow and adult cDNA data, mpiBLAST was the most efficient one, and the total run time of the Hadoop program and the serial

program were close to each other and were about twice as slow as the mpiBLAST. This result indicated that for data sets at this scale, Hadoop's efficiency gain from utilizing the MapReduce algorithm and HDFS still did not overcome the existence of latency time. mpiBLAST, on the other hand, handled this scale of data very well.

For the large scale human normal and cancer dataset, mpiBLAST was still the most efficient program, although the advantages of the MapReduce algorithm started to manifest and the total run time of the Hadoop program became very close to that of mpiBLAST. On the other hand, with this scale of data, the serial NCBI BLAST required much longer to accomplish the calculation.

It was worth noting that in the new Hadoop programs, the total run time of the job also depended on the chunk size of the input data. The chunk size values used in this comparison was 500, 800, 2000 and 2000, for small, median, large and huge scale datasets, respectively. The effects of chunk size are discussed in more details in the next chapter (Section 5.1.3).

For the huge scale artificial data set, the program with the best performance was the Hadoop Sequence Comparison tool. The Hadoop program was more efficient than mpiBLAST for the following reasons:

- When handling data sets with this scale, the latency time of the Hadoop program was negligible compared to the actual calculation time.
- The Hadoop program was fully automated. In other words, both of the two time consuming steps, sequence query (in mapper 2) and sequence-ID matching (in mapper 3 and reducer 3) were parallelized using the MapReduce algorithm. On the other hand, mpiBLAST only parallelized the sequence query step. The sequence-ID matching step, who required polynomial runtime, ran in the serial mode.
- Comparing with the MPI strategy, the Hadoop software framework and the MapReduce algorithm scaled better with the size of the data, the scalability performance is discussed below.

It was observed that the other two Hadoop programs, the Digital DNA Library Subtraction tool and the Sequence De-duplication tool presented very similar run time when the same testing dataset were processed, which was a reasonable result, considering the three programs only differ on the very last stage of the calculation.

4.3.3 Scalability

As discussed in Section 2.1, with the decreased cost of obtaining reliable raw data, the scales of Bioinformatics projects are constantly growing. Nowadays, Bioinformaticians need to deal with datasets up to large gigabytes scale very frequently (Pennisi, 2010). As a result, scalability becomes an important property to measure the success of a Bioinformatics program.

4.3.3.1. Testing Program and Data

In order to measure the scalability of the new Hadoop programs, the sequence similarities of the four data sets, the small scale next generation sequencing data, the median scale Fathead minnow and adult cDNA data, the large scale human normal and breast cancer data, and the huge scale artificial data were processed using NCBI BLAST, mpiBLAST, and the new Hadoop Sequence Similarity Comparison tool respectively. The chose of testing datasets was a balance between the experimental design and the available computational resources.

4.3.3.2. Results

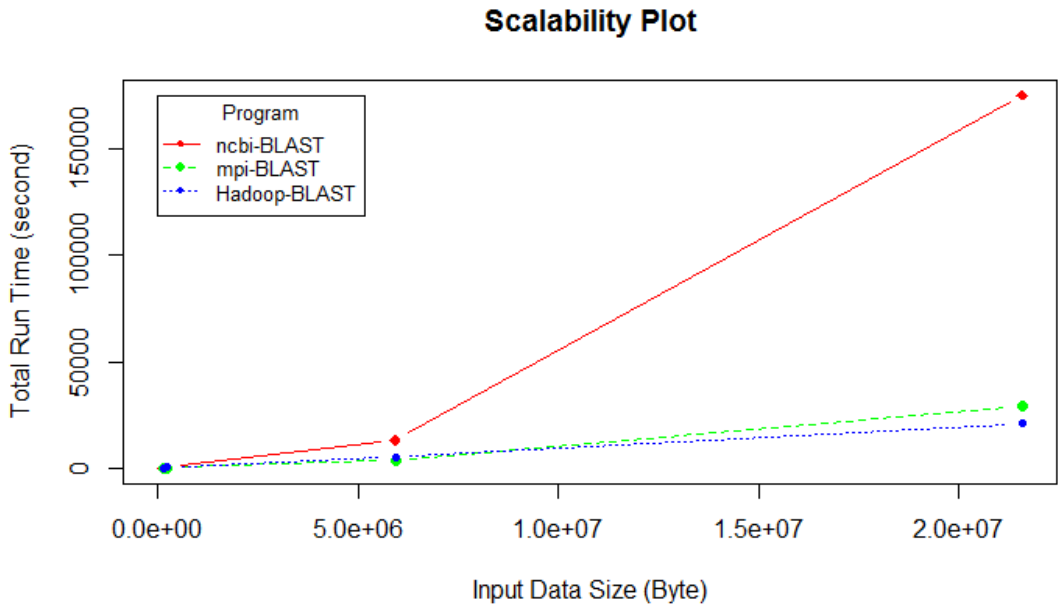

Figure 4.18. The Scalability Plots of NCBI BLAST, mpiBLAST and Hadoop Sequence Comparison Tool

Table 4.1 showed the detailed run time of each case. The scatter plots shown in Figure 4.18 and Figure 4.19 illustrate the scalabilities of different programs in a more straightforward way.

Figure 4.18 plots the run time of each program against the size of the input testing data set. From this plot, it can be concluded that the scalability of NCBI BLAST was significantly worse than the scalabilities of mpiBLAST and the new Hadoop program, especially when the size of the data was large. It was worth noting that the documented run time (48 hours 34 minutes) came from the only successful job among several attempts. All other attempts failed due to various reasons, indicating the extremely poor scalability of the serial program.

Figure 4.19 shows a zoom-in plot of the scalabilities of the two parallelized programs, mpiBLAST and the Hadoop Sequence Comparison tool. From this plot, the researcher reached the conclusion that although mpiBLAST outperformanced

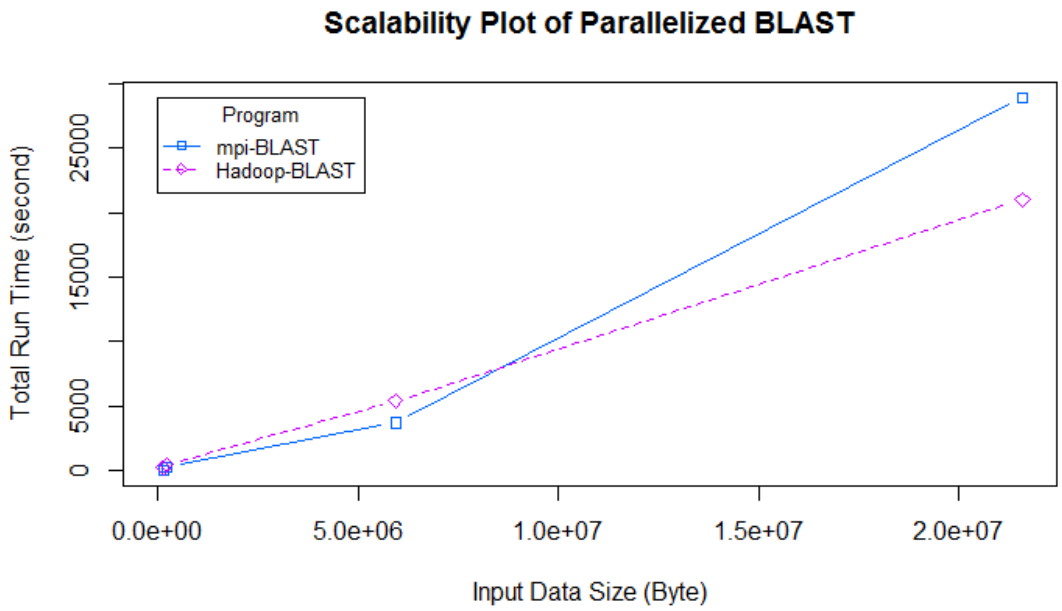

Figure 4.19. The Zoom In Scalability Plots of mpiBLAST and Hadoop Sequence Comparison Tool

the new Hadoop program when the size of the input files were relatively small, the scalability advantage of the Hadoop Sequence Comparison tool was obvious when extra large scale calculations were in need. For the Hadoop version of the Sequence Similarity Comparison tool, although the scalability plot was not exactly linear, a good scalability performance was observed.

4.3.4 Conclusion

In conclusion, the above measure of successfulness results were mostly consistent with prior literature reviews and the researcher's expectations that the MapReduce parallelization strategy may not always have as much efficiency gain as MPI parallelization strategy, especially with small to median scale datasets; However, comparing with the serial implementations, the improvement of total run

time was significant, and better scalability over the MPI implementation was also observed with extra large scale data processing.

4.4 Summary

This chapter described the results of this project from three different aspects: the final multi-step MapReduce algorithms that were implemented into the three new Hadoop programs, the design of the graphical user interface, and the measure of successfulness of the project. From the current results, which were consistent with the researcher's initial expectations, the proposed goals were achieved.

In the next chapter, some technique details, design considerations, and valuable findings are discussed.

CHAPTER 5. DISCUSSIONS AND FINDINGS

This project evaluated the feasibility and potential advantages of utilizing Hadoop's MapReduce algorithms and HDFS in the design and implementation of various Bioinformatics programs. The important findings and technique details that emerged from the design and implementation phases of the three new Bioinformatics tools (the DNA Sequence Similarity Comparison tool, the Digital DNA Library Subtraction tool, and the Sequence De-duplication) and some additional considerations in the design of the graphical user interface are presented in this chapter.

5.1 The Design and Implementation of Mappers and Reducers

The final implementations of the MapReduce algorithm in the three new Bioinformatics tools are discussed in Section 4.1. This algorithm was somewhat counterintuitive and not exactly the same as the researcher's original proposal. The reason of the change, and the considerations and indications of the new design are discussed below.

5.1.1 The Original Design and the Issues

Instead of the current algorithm with one pre-processing step, three mappers, two reducers, and one post-processing step, the original design only included one MapReduce step besides the pre- and post-processing steps. The original design was very intuitive. After all input sequences were rewritten into the one-line key-value pair format, and the library tag added to the key in the pre-processing step, a BLAST call was made on each key-value pairs, based on the name of the library

embedded in the key, and the sequence string included in the value. The input key-value pairs of the mapper had the following format:

```
lib>ID    sequence
```

where "lib>ID" was the key and "sequence" was the value. If the BLAST call returned a match with e-value or percentage similarity beyond the user defined cutoff, than a status flag "shared" was appended to the key, otherwise, the status flag "unique" was appended. The output key-value pair had the following format:

```
lib|||status    >ID|||sequence
```

where "lib|||status" was the key, ">ID|||sequence" was the value, and "status" was either "shared" or "unique".

The reducers of the three Hadoop based Bioinformatic tools, the DNA/protein Sequence Similarity Comparison tool, the Digital DNA Library Subtraction tool, and the Sequence De-duplication tool were slightly different. They were designed based on the comparisons each program desired to conduct. Furthermore, in the reduce step, the input key-value pairs would have only four possible keys. If the input sequence files were named seq1.fasta and seq2.fasta, then the possible input keys of the reducer step included:

- seq1|||unique
- seq2|||unique
- seq1|||shared
- seq2|||shared

The reducer simply combined the IDs and sequences with the same key into the multi-line FASTA/FASTQ format, and reformatted the input keys so that they were consistent with the names of the output files. The output key-value pairs of the reducer were then downloaded from HDFS to local file system, and split into multiple files in the post-processing step.

The above design was intuitive, highly parallelized and took full advantage of the MapReduce algorithm and HDFS; however, it was not efficient. The researcher analyzed the run time of each line of code in the mapper and the reducer, and found that this issue was generated from the way the BLAST command was executed. In each BLAST call, the indexed library was first loaded to memory, then the search was executed by taking the query sequences from the standard input. For example, if there were a total of n sequences in the two input files, and their sequence similarities were calculated by using the above fully parallelized algorithm, then the library needed to be loaded for a total of n times, which significantly jeopardized the performance of the program.

5.1.2 The New Design

In order to overcome the above issue, the new MapReduce algorithm that is described in Section 4.1 was designed and implemented. In this algorithm, the input files were first split into chunks by mapper 1 and reducer 1, and then each chunk invoked one BLAST call. In this design, the number of times needed to load the indexed library was reduced significantly. This led to not only increased calculation efficiency, but also reduced I/O. It was also observed that the performance of the new MapReduce algorithm depended heavily on the choose of chunk size. The effects of chunk size are discussed below.

5.1.3 The Effects of Chunk Size

In the three new Bioinformatics tools, the chunk size was defined as the number of sequences in each input key-value pair in the mapper 2 step. When deciding the chunk size, the following factors needed to be considered.

- The relative chunk size.

  - If the assigned chunk size were too small compared to the size of input data, the number of times the library needed to be loaded would still be high, which led to suboptimal performance.
  - If the assigned chunk size were too large comparing with the size of input data, then only a few partitions of the input data would be generated, which also led to suboptimal performance. In other words, the program was not fully parallelized and did not take full advantage of the available computer resources.

- The absolute chunk size.

  The absolute size of each chunk was also an import factor to consider. In Hadoop software framework, input data is replicated over data nodes, as a result, the size of each chunk should not exceed the capacity of each node.

- The total number of mappers running at the same time.

  Because of the need to load libraries into memory, if too many mapper jobs were running simultaneously, the required memory may very likely exceed the total available memory. In order to avoid this issue, a maximum number of mappers was restricted to 20. This was also related to chunk size. Increasing the number of sequences of each chunk, or reducing the number of chunks would decrease the number of mapper jobs and thus decrease the memory usage. However, as described above, the absolute chunk size should not exceed the capacity of each node.

In order to investigate how the chunk size influenced the performance of the Hadoop programs, the following test was executed. The Hadoop Sequence Similarity Comparison tool and the median scale Fathead minnow and adult cDNA data were used in the test. There were a total of 262,638 sequences in the two input files. The test runs were conducted with chunk size 1, 50, 100, 200, 400, 800, 1200, 2000, 3000, 4000, 5000 and 10000, respectively. The run time of each test run was documented and Figure 5.1 shows the plot of program run time of different chunck size.

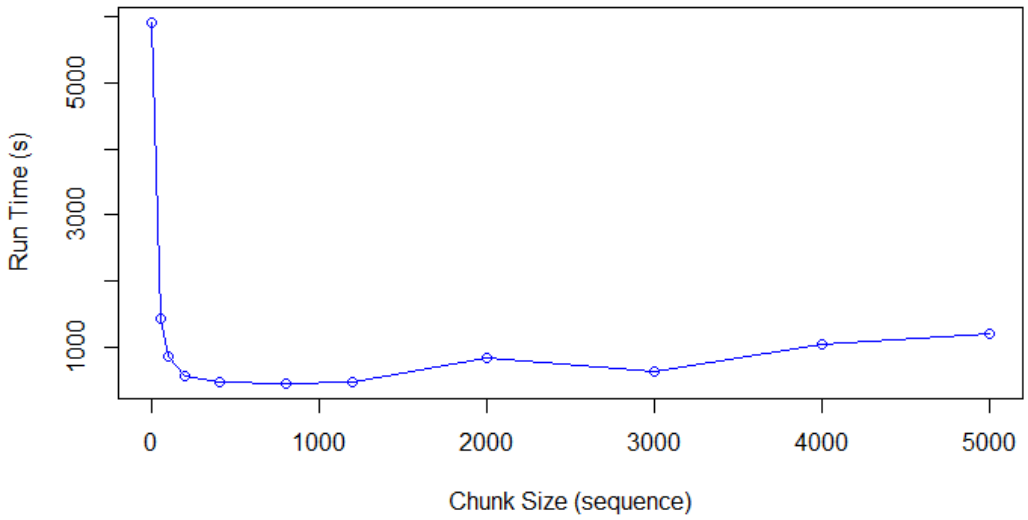

Figure 5.1. The Effect of Chunk Size on the Performance of Hadoop Sequence Similarity Comparison Tool.

The above figure clearly shows that with chunk size = 1, in other words, one BLAST call was invoked by each input sequence, the performance of the program was very bad. With the increase of the chunk size, the program ran more and more efficiently. When the chunk size was between 200 and 1200 sequences, the performance was relatively stable, and then with the further increase of the chunk size, the run time gradually increased again. In the previous discussion of program efficiency (Section 4.3), the run time with chunk size = 800 was used for this median scale dataset. In addition, for the small, large and huge scale datasets, chunk size values of 500, 2000 and 2000 were used, respectively. This result provided several important indications:

- There was an optimal window, instead of just an optimal value of the chunk size. In other words, when the chunk size fell within a certain range, the

performance of the program was relatively stable. This characteristic provided users with some flexibilities in defining the chunk size.

- When the chunk size was above 5000, the program crashed very often because of memory issue, indicating that a very large chunk size would not be recommended.
- The optimal window of the chunk size depended on the size of the input data. For example, the researcher found that when using small scale and large scale testing data set, the optimal chunk size varied a lot.

In conclusion, the partition of input data into chunks, instead of calling the BLAST function for each sequence greatly improved the efficiency of the program. On the other hand, the impact of the size of each chunk on the overall performance of the program cannot be ignored. There existed an optimal window of chunk size, which depended heavily on the scale of input data.

5.1.4 Other Technical Details

In the design and implementation of the MapReduce algorithm, there emerged a couple of additional technical details that the researcher believed were worth further discussing.

5.1.4.1. The Debug of Mappers and Reducers

The debugging of Python mappers and reducers under the Hadoop Streaming environment had been reported to be difficult, and the researcher's experience was consistent with that. Hadoop was developed in Java, and only the general Java error messages are provided when a Hadoop Streaming job fails. This general Java error message includes information such as the input and output files, total number of mappers and reducers, and whether the error occurs in the mapper

step or the reducer step, which is useful, but not informative enough if the error comes from a program bug in the Python scripts. In order to overcome this difficulty, the following strategies were attempted by the researcher.

- Debugging the Python mappers and reducers in local file system with a small scale testing dataset before executing them using Hadoop Streaming.

  By default, Hadoop Streaming takes input from the standard input and prints output to the standard output, and the output key-value pairs of the mapper are shuffled and sorted by key before they are passed into the reducer. As a result, the MapReduce environment could be mimicked by piping the input file, the mapper, the sort step and the reducer together in a Linux command line environment.

  For example, if the mapper was called "mapper.py", with two arguments, -a, which took a integer parameter, and -f, which was a flag argument; the reducer was called "reducer.py", which took no argument; and the input file was called input.txt and was stored under the same directory as the mapper and the reducer, then the Python mapper and the reducer could be tested using the following bash command:

  ```
  cat input.txt | python mapper.py -a 2 -f | sort |\
  python reducer.py
  ```

  If program bugs existed in the mapper.py or reducer.py above, then detailed Python error messages, which indicated the line number with bugs, as well as the error type were given.

  Although this method is a quick way to debug mappers and reducers, there are some limitations:

  - This method only checks the syntax correctness of mappers and reducers. In other words, whether the script is compilable by the current Python

compiler. However, not all compilable Python scripts can be executed as mappers or reducers. An additional requirement was to be compatible with the MapReduce Framework. Namely, the transformations in mappers should be mappable to all input units (key-value pairs), which should be independent of each other, and the actions in reducers should only depend on different key values.

- One should also keep in mind that the versions of Python compilers could be different between the local file system and the Hadoop system. As a result, it would be better if the mappers and reducers were written in a general manner and were compilable by different Python versions. For example, if "input" were a string variable, the following two lines of code would print exactly the same output in Python 2.7.0 and later versions. However, the first line is not recognized by versions of Python prior to 2.7.0, and thus is not as general as the second line.

```
print "Input file: {}".format(input)
print "Input file: {0}".format(input)
```

- If a small subset of the real input dataset was used in the local testing, one should also keep in mind that the full dataset may not be as clean as the testing data, and that more exceptions could exist. On strategy to deal with this issue was to adopt proper exception handling mechanisms, which is discussed in below. The other strategy was to write mappers and reducers in the most rigorous way. For example, if "key", "value" and "line" were three string variables, and "line" was supposed to contains a single tab, then the following two lines of code did exactly the same variable assignments. They split "line" based on the tab, and assigned the first half to variable "key" and the second half to variable "value". However, if one could not guarantee that every line in the file had one single tab, then the second line would be a better way to write

the code, because it prevented run time errors that could be generated if tabs existed in the value part of the line.

```
key, value = line.split("\t")
key, value = line.split("\t", 1)
```

- Even if all the mappers and reducers were able to compile in local system, in order to guarantee their appropriate and successful execution in the Hadoop system, all rules about the generic and streaming options of Hadoop Streaming should be followed. Some of the options were not so straight forward and intuitive. Those are discussed in Section 5.1.4.3.

- Introducing exception handling procedures whenever possible.

  As discussed above, no one could guarantee that all possible situations had been considered and included in mappers and reducers, especially when the size of input data was large. As a result, the introduction of exception handling mechanisms in mappers and reducers were very important to assure the execution of the MapReduce job without any runtime error.

- Debugging the mappers and reducers piece by piece.

  If the above two precautions had been conducted and the MapReduce job still exited with error status, then it became very difficult to locate the issue. What the researcher did was breaking down the complex script to shorter modules and then identifying the problematic module. The first step to break down the complex structure was to run the mapper and the reducer separately. In Hadoop Streaming environment, this could be done by specifying the following general option:

  ```
  -D mapreduce.job.reduces=0
  ```

  After pinpointing the script that caused the problem, one could add break point to the script at a proper position. If the shortened version of the script

was executed in the Hadoop Streaming environment with no runtime error, then, the break point could be moved down, until the location of the issue was identified.

5.1.4.2. The Choose of Delimiters

As discussed in chapter 4, the researcher introduced several uncommon delimiters in the MapReduce algorithm. For example, the "|||" delimiter in mapper 1, "__" in reducer 1, "|??|" in mapper 2 and so on. The purpose of using those uncommon and complex delimiters was to keep the integrity of the program to the largest extent. For example, the "|||" delimiter in mapper 1 separated the unique ID and the sequence of one input molecule. By using this uncommon delimiter instead of common symbols like "|", the researcher attempted to avoid situations where "|" appeared in the unique ID itself, and thus in the following steps, the ID and sequence could be split in an inappropriate location.

It had been realized that this strategy could not avoid all improper parsing, but the use of the uncommon delimiters would certainly decrease such chances.

5.1.4.3. The Generic and Streaming Options of Hadoop Streaming Command

The Hadoop Streaming is a utility that came with the Hadoop distribution. The utility allows the users to create and run MapReduce jobs with any executable or script as the mapper and/or the reducer. It dramatically decreases the programming complexity of Hadoop by allowing the usage of any programming languages. However, this flexibility comes with the sacrifice of the simplicity of the Hadoop command. For example, below were the command the researcher used to

execute the mapper 3 and the reducer 3 of the Hadoop Digital DNA Library Subtraction tool:

```
hadoop jar /usr/lib/gphd/hadoop-mapreduce/hadoop-streaming.jar \
    -D mapred.reduce.tasks=2  \
    -input input1/kv.txt \
    -output hadoop-blast/output/output1/mr3  \
    -mapper "mapReduce_code/mapper3_sub.py -i output1"  \
    -file mapReduce_code/mapper3_sub.py \
    -file output1_idlist.pkl  \
    -reducer "mapReduce_code/reducer3.py -a fish -b fry"  \
    -file mapReduce_code/reducer3.py
```

Although those streaming commands were coded into the GUI and were automatically generated for the users, some of the underlying rules were complex and needed to be explained.

Generally speaking, there are two kinds of options in a Hadoop Streaming command, the generic options and the streaming options. Generic options specifies Hadoop environment parameters and configuration variables. For example, the maximum number of mappers, the number of replicates to be made, the location of the application configuration files, and so on. The streaming options on the other hand specifies parameters related to the current MapReduce job. For example, the locations and names of the input and the output files in the HDFS, the locations and names of the mapper and reducer executables in the local file system, the format of the input/output key-value pairs and so on.

Taking the above Streaming command as an example, the first line of the command specified the Hadoop Streaming executable file. The second line was a generic option that specified that two reducers were required. The third and fourth lines specified the HDFS location of the input and the output key-value pairs. The fifth and eighth lines called the mapper and the reducer executables respectively,

with all necessary command line arguments. The sixth, seventh, and ninth lines specified the names and locations of the files to be copied from the local file system to the Hadoop clusters.

Listed below were some rules that were worth noting. Most of them came from the research's experience of running failed MapReduce commands.

- The generic options must be placed before the streaming options.
- The mappers and the reducers must be made readable and executable by all users before the execution of the command. In other words, the following command that changed the privileges should be executed:

  ```
  chmod a+rx  *.py
  ```

  The mappers and reducers have to be copied from the local system to HDFS, which means the HDFS need the read privilege of the files in the local system. In addition, the Hadoop Streaming API requires that the mappers and reducers must be called as executables, indicating the need of execute privileges of the files.

- The following line needed to be written as the header of all mappers and reducers:

  ```
  #! /usr/bin/python
  ```

  As discussed above, all mappers and reducers needed to be executable, as a result, the above line was needed.

- When executing mappers and reducers, if any command line arguments were needed, the name of the executable file as well as the arguments needed to be quoted.
- Extra caution needed to be taken if "-file" streaming option was used. The "-file" option could pass two kinds of files:

- The mapper and the reducer executable files.
- Files that were required when the mapper and/or the reducer were executed. For example, pickle files, lookup tables, and so on.

For the first kind of files, the values of the "-file" option could contain the file name and any relative or absolute paths. However, for the second kind of files, they have to be located in the current directory, and the value of the "-file" option should only contain the name of the file.

5.1.5 Conclusion

In conclusion, this section described the considerations and important findings in the design and implementation phases of the MapReduce algorithm in the newly developed Hadoop Bioinformatics tools, the Sequence Similarity Comparison tool, the Digital DNA Library Subtraction tool, and the Sequence De-duplication tool. In addition, the effects of chunk size on the performance of the programs were discussed. Also presented were some valuable technique details related to the design and implementation of the mappers and the reducers.

5.2 The Graphical User Interface

As discussed in Section 4.2, a graphic user interface was created to facilitate the easy usage of the new Hadoop Bioinformatics toolkit. In this section, the back end settings that enabled multi-user access and some user friendly features were presented.

5.2.1 The Multi-User Setting

Any software would not be useful if at a given time, only one user has access to it. In order to guarantee that multiple users and multiple processes could access

the above Hadoop tools smoothly and simultaneously, the following strategies had been taken.

- A unique directory was created for each job.

  The name of the first input file without the suffix, the tool that was executed, and the user specified chunk size were concatenated together and used as the unique directory name. For example, if the first input file was called breast_cancer.fasta, the Sequence De-duplication tool was executed with a chunk size of 1000, then the new directory that all intermediate and output files were stored would be named "breast_cancer_dedup_1000". This directory would be created in both the local file system and the HDFS.

  If the user input created a directory name that already existed in the local file system, then the GUI would automatically add a suffix to the current name to make it unique. For example, if the execution of the job in the above example created a directory called "breast_cancer_dedup_1000", the next time if a job with the same setting was executed, a new directory named "breast_cancer_dedup_1000_1" would be created.

  The use of unique directory names guaranteed that multiple jobs did not interfere with each other if they were running at the same time, and that files generated by previous users were not removed by later users accidentally. In the future, if this toolkit were exposed to larger range of users, the use of unique section IDs as the name of directories might be a better choice.

- All intermediate files were deleted after each job.

  After each job, the GUI cleaned up the local file system by removing all the files except the final output sequence files, the log file, and the error file. It also cleaned the HDFS, simply by removing the whole directory. This step not only released the disk space, but also simplified the file structure, so that the output files became easier to find by the users.

- Absolute paths were used in the GUI, the mappers and the reducers.

  Currently, the GUI, the mappers and the reducers are callable by any users who have access to Hathi (hathi.rcac.purdue.edu). In order to ensure that the programs could run properly from any location of the file system, the absolute paths were enforced in both the front end GUI, and the back end mappers, reducers, pre-processing scripts and post-processing script.

- All users only have read/write privilege to their own output files.

  In order to guarantee that all users had enough computer resource to run jobs and store necessary data, all intermediate and output files were generated and preserved under the researcher's space. In order to ensure that the end users had access to, and only to data generated by their own jobs, read/write privileges were granted to the owner of the file as well as the researcher, who might need to purge the system from time to time.

5.2.2 The User Friendly Features

Prior to this project, the researcher studied the usability and user experience situations in Bioinformatics software (Wang & Springer, 2015). Some of the findings in that study were applied in the design of the current GUI. Specifically, the following user friendly features were included.

- Alternative sources of information were provided.

  According to the researcher's prior research (Wang & Springer, 2015), the redundancy of information was very crucial for the users to understand when and how to use a Bioinformatics software. As a result, two major sources of information, the introductory pages (see Figure 4.4, Figure 4.5 and Figure 4.6 for details) and a separate document were provided. The following details were included in the two sources.

  - The theory and the purpose of each program.
  - The transformation of data the tool would carry out.
  - The expected format of the input and output data.
  - The default values of the parameters.
  - The methods to optimize performance.
  - Examples from real use cases of each program.

  Most of the details were repeated or described from different perspectives in the two sources. For space limitations, the examples were only included in the document.

- Log and error files were provided.

  Detailed log files and error files were made accessible to GUI users. For successful jobs, the log and error files provided information about the total run time, the parameter of each step, such as E-values, the chunk size, the molecular type, and so on. Also included were the Hadoop Streaming configuration parameters, such as the number of mappers and reducers executed, the status of the streaming jobs, the memory usage, and so on. For failure jobs, the log and error file provided valuable information for users to change their input and conduct the calculation pipeline in a proper and reasonable manner.

- User testing.

  Currently, user testing had not yet been conducted. If the programs were to be released to larger range of users, user testing would be carried out to collect feedback from potential users, and modifies would be made accordingly.

5.3 The Findings from the Methodology Study

As mentioned in Section 1.2, besides the design and implementation of the front end GUI and the back end MapReduce algorithm, another important aim of this project was to investigate the Hadoop transformation of Bioinformatics software as a methodology study. Based on this specific aim, the following findings were discovered.

- The programming complexity.

  One of the characteristics of Hadoop software framework is the compatibility of all programming languages in mappers and reducers, which was often reported as an important advantage of Hadoop, because it decreased the programming complexity of parallelizing a serial program. According to the research's experience, although it is true that the concepts of mappers and reducers are straightforward, and various programming languages could be used to implement the MapReduce algorithm, it should also be noted that the debugging of mappers and reducers were challenging and extremely painful if Hadoop Streaming was used. In addition, in order to optimize the performance of a MapReduce job, experience and background knowledge were in great need. In other words, if one wanted to parallelize any serial code with real usage to its Hadoop version, the complexity was still high, although compared to other parallelization strategies, such as MPI and OpenMP, Hadoop might be easier initially.

- The advantages of using Hadoop in Bioinformatics.

  The great scalability performance of the Hadoop software framework, which had been established by this project as well as prior publications, was an advantage of using Hadoop in Bioinformatics, because nowadays, Bioinformatics projects often involved large to ultra-large scale data manipulations and calculations. Besides that, the fault tolerance file system, and the capabilities of handling of low level partitioning details by the

framework were all advantages of using Hadoop in the design and implementation of other Bioinformatics software.

- The disadvantages of using Hadoop in Bioinformatics.

  All coins have two sides. With the above mentioned advantages, one should also keep the following disadvantages in mind.

  - Large latency time.

    As discussed in Section 4.1, the latency time takes a large percentage of the total run time when the size of the input data was not so large. In other words, with small to median scale data, the Hadoop version of a program could be much slower than other parallelization strategies, or even its serial counterpart.

  - Programming complexity.

    The programming complexities of Hadoop's mappers and reducers were not as low as it had been reported, and debugging and optimizing the performance could be especially challenging.

  - High hardware requirements.

    Hadoop replicates input files to ensure the false tolerant executions of the MapReduce job. Moreover, because of the manner mappers and reducers are executed under the Hadoop framework, high volumes of communications between nodes are expected. In other words, computer nodes with large disk space and high I/O capacity are required by Hadoop clusters, which could be challenging under some circumstances.

5.4 Future Direction

The current Hadoop cluster, Hathi, has its limitations. For example, it is a shared memory cluster, and as a community cluster, it has relatively strict

limitations on memory and quota. To further study Hadoop as a parallelization strategy to improve Bioinformatics programs, generalize and potentially extrapolate the current results, other more repeatable and accessible platforms, such as cloud computing systems, might be investigated.

In addition, a quantitative study of the chunk size may enable further optimization of the current three Hadoop Bioinformatics programs. Regarding the design of the GUI interface, if the programs were to be released to larger range of users, user testing would be carried out to collect feedback from potential users, and modifies would be made accordingly.

5.5 Conclusion

In conclusion, this chapter discussed the important findings, technical details and additional considerations in the design and implementation of the front end graphical user interface and the back end MapReduce algorithm of the three new Hadoop Bioinformatics programs, the Sequence Similarity Comparison tool, the Digital DNA Library Subtraction tool, and the Sequence De-duplication tool. Also included were the findings of the methodology study.

CHAPTER 6. SUMMARY

The project described in this dissertation attempted to improve the efficiency and scalability performance, as well as the usability of three Bioinformatics applications: DNA/peptide sequence similarity comparison, digital DNA library subtraction, and DNA/peptide sequence de-duplication by 1) adopting the Hadoop MapReduce algorithms and distributed file system and 2) implementing the fully automated Hadoop programs into user friendly graphical user interface. In addition, the researcher was also interested in investigating the advantages and limitations of the application of the Hadoop software framework as a general method in optimizing Bioinformatics programs.

After considering the original calculation algorithms in the serial version of the programs, the available computational resources, the nature of the MapReduce framework, and the optimization of performance, a processing pipeline with one pre-processing step, three mappers, two reducers and one post-processing step was developed. Then a GUI interface that enabled users to specify input/output files and program parameters was created. Also implanted into the GUI were user friendly features such as organized instruction, detailed log files, multi-user accessibility, and so on.

The new and fully automated Bioinformatics toolkit showed execution efficiency compatible with their MPI counterparts with median to large scale data, and better efficiency than MPI when ultra-large dataset was provided. In addition, good scalability was observed with testing dataset up to 20 Gb.

The methodology study showed that although not a cure-all, Hadoop software framework was generally a good candidate from the parallelization of Bioinformatics programs due to its low programming complexity, easy-to-understand MapReduce concepts, good scalability and other features.

LIST OF REFERENCES

LIST OF REFERENCES

Ahmadian, A., Gharizadeh, B., Gustafsson, A., Sterky, F., Nyrén, P., Uhlén, M., & Lundeberg, J. (2000). Single-nucleotide polymorphism analysis by pyrosequencing. *Analytical biochemistry*, *280*(1), 103–110.

Akalin, P. K. (2006). Introduction to bioinformatics. *Molecular Nutrition & Food Research*, *50*(7), 610-619. doi: DOI10.1002/mnfr.200500273

Altschul, S., Madden, T., Schäffer, A., Zhang, J., Zhang, Z., Miller, W., & Lipman, D. (1997). Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, *25*(17), 3389–3402.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, *215*(3), 403-410. doi: DOI10.1006/jmbi.1990.9999

Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., Brown, C. G., ... others (2008). Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, *456*(7218), 53-59. doi: Doi10.1038/Nature07517

Bjornson, R. D., Carriero, N. J., Colangelo, C., Shifinan, M., Cheung, K. H., Miller, P. L., & Williams, K. (2008). X!!tandem, an improved method for running x!tandem in parallel on collections of commodity computers. *Journal of Proteome Research*, *7*(1), 293-299. doi: Doi10.1021/Pr0701198

Boratyn, G., Schäffer, A., Agarwala, R., Altschul, S., Lipman, D., & Madden, T. (2012). Domain enhanced lookup time accelerated blast. *Biol Direct*, *7*(1), 12.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, *26*(2), 4.

Chen, C., He, M., Zhu, Y., Shi, L., & Wang, X. (2015). Five critical elements to ensure the precision medicine. *Cancer and Metastasis Reviews*, *34*(2), 313-318.

Cohen, J. (2004). Bioinformatics–an introduction for computer scientists. *ACM Computing Surveys*, *36*(2), 122-158.

Cravatt, B., Simon, G., & Yates III, J. (2007). The biological impact of mass-spectrometry-based proteomics. *Nature*, *450*(7172), 991–1000.

Darling, A. E., Carey, L., & Feng, W. (2003). The design, implementation, and evaluation of mpiblast [Conference Proceedings]. In *4th international conference on linux clusters: The hpc revolution 2003 in conjunction with clusterworld conference & expo* (p. 13-15).

Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107-113.

Diatchenko, L., Lau, Y. F., Campbell, A. P., Chenchik, A., Moqadam, F., Huang, B., ... Siebert, P. D. (1996). Suppression subtractive hybridization: a method for generating differentially regulated or tissue-specific cdna probes and libraries. *Proc Natl Acad Sci U S A*, *93*(12), 6025-30.

Drmanac, R., Sparks, A. B., Callow, M. J., Halpern, A. L., Burns, N. L., Kermani, B. G., ... Reid, C. A. (2010). Human genome sequencing using unchained base reads on self-assembling dna nanoarrays. *Science*, *327*(5961), 78-81. doi: DOI10.1126/science.1181498

Fishman, G. (2013). *Monte carlo: concepts, algorithms, and applications*. Springer Science & Business Media.

Gabriel, E., Fagg, G., Bosilca, G., Angskun, T., Dongarra, J., Squyres, J., ... Woodall, T. (2004, September). Mpi: Goals, concept, and design of a next generation mpi implementation. In *Proceedings, 11th european pvm/mpi users' group meeting* (pp. 97–104). Budapest, Hungary.

Gorlatch, S., & Bischof, H. (1998, DEC). A generic mpi implementation for a data-parallel skeleton: Formal derivation and application to fft. *Parallel Processing Letters*, *8*(4), 447-458.

Heller, M. J. (2002). Dna microarray technology: Devices, systems, and applications. *Annual Review of Biomedical Engineering*, *4*, 129-153. doi: DOI10.1146/annurev.bioeng.4.020702.153438

Houghton, R. L., Dillion, D. C., Molesh, D. A., Zehentner, B. K., Xu, J. C., Jiang, J., ... Reed, S. G. (2001). Transcriptional complementarity in breast cancer: Application to detection of circulating tumor cells. *Molecular Diagnosis*, *6*(2), 79-91. doi: Doi10.2165/00066982-200106020-00003

Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., & McVean, G. (2012). De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature Genetics*, *44*(2), 226-232. doi: Doi10.1038/Ng.1028

Jiang, Y. Q., Harlocker, S. L., Molesh, D. A., Dillon, D. C., Stolk, J. A., Houghton, R. L., ... Xu, J. C. (2002). Discovery of differentially expressed genes in human breast cancer using subtracted cdna libraries and cdna microarrays. *Oncogene*, *21*(14), 2270-2282. doi: DOI10.1038/sj/onc/1205278

Kane, M. D., Springer, J. A., Iannotti, N. V., Gough, E., Johns, S. M., Schlueter, S. D., & Sepulveda, M. S. (2008). Identification of develpment and tissuespecific gene expression in the fathead minnow pimephales promelas, rafinesque using computational and dna microarray methods. *Journal of Fish Biology*, *72*(9), 2341-2353.

Koonin, E. V. (2005). Orthologs, paralogs, and evolutionary genomics. *Annual Review of Genetics*, *39*, 309-338. doi: DOI10.1146/annurev.genet.39.073003.114725

Langmead, B., & Salzberg, S. (2012). Fast gapped-read alignment with bowtie 2. *Nature methods*, *9*(4), 357–359.

Langmead, B., Schatz, M. C., Lin, J., Pop, M., & Salzberg, S. L. (2009). Searching for snps with cloud computing. *Genome Biol*, *10*(11), R134. Retrieved from `http://www.ncbi.nlm.nih.gov/pubmed/19930550` doi: 10.1186/gb-2009-10-11-r134

Leo, M., Santoni, F., & Zanetti, G. (2009). Biodoop: Bioinformatics on hadoop. *International Conference on Parallel Processing Workshops*, 415-422. doi: 10.1109/ICPPW.2009.37

Lewis, S., Csordas, A., Killcoyne, S., Hermjakob, H., Hoopmann, M. R., Moritz, R. L., ... Boyle, J. (2012). Hydra: a scalable proteomic search engine which utilizes the hadoop distributed computing framework. *BMC Bioinformatics*, *13*, 324. doi: 10.1186/1471-2105-13-324

Li, R., Li, Y., Fang, X., Yang, H., Wang, J., Kristiansen, K., & Wang, J. (2009). Snp detection for massively parallel whole-genome resequencing. *Genome research*, *19*(6), 1124–1132.

Lin, H., Ma, X., Feng, W., & Samatova, N. (2011). Coordinating computation and i/o in massively parallel sequence search. *Parallel and Distributed Systems, IEEE Transactions on*, *22*(4), 529–543.

Liu, S., Chen, X., Yan, Z., Qin, S., Xu, J., Lin, J., ... Shui, W. (2014). Exploring skyline for both mse-based label-free proteomics and hrms quantitation of small molecules. *Proteomics*, *14*(2-3), 169–180.

Lowry, R. (2014). Concepts and applications of inferential statistics.

Macaulay, C., Sloan, D., Jiang, X. Y., Forbes, P., Loynton, S., Swedlow, J. R., & Gregor, P. (2009). Usability and user-centered design in scientific software development. *Ieee Software*, *26*(1), 96-102.

Madden, T. (2013). The blast sequence analysis tool.

Mardis, E. (2008). The impact of next-generation sequencing technology on genetics. *Trends in genetics*, *24*(3), 133–141.

Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., ... Rothberg, J. M. (2005). Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, *437*(7057), 376-380. doi: Doi10.1038/Nature03959

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., ... DePristo, M. A. (2010). The genome analysis toolkit: A mapreduce framework for analyzing next-generation dna sequencing data. *Genome Research*, *20*(9), 1297-1303. doi: DOI10.1101/gr.107524.110

McKernan, K. J., Peckham, H. E., Costa, G. L., McLaughlin, S. F., Fu, Y. T., Tsung, E. F., ... Blanchard, A. P. (2009). Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding. *Genome Research*, *19*(9), 1527-1541. doi: DOI10.1101/gr.091868.109

Niemenmaa, M., Kallio, A., Schumacher, A., Klemela, P., Korpelainen, E., & Heljanko, K. (2012). Hadoop-bam: directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, *28*(6), 876-7. doi: 10.1093/bioinformatics/bts054

NIH-National-Human-Genome-Research-Institue. (2015). *The human genome project completion: Frequently asked questions.* Retrieved from `http://www.genome.gov/11006943`

Nordberg, H., Bhatia, K., Wang, K., & Wang, Z. (2013). Biopig: a hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics*, *29*(23), 3014-9. doi: 10.1093/bioinformatics/btt528

O'Driscoll, A., Daugelaite, J., & Sleator, R. D. (2013). 'big data', hadoop and cloud computing in genomics. *Journal of Biomedical Informatics*, *46*(5), 774-781. doi: DOI10.1016/j.jbi.2013.07.001

Pearson, W., & Miller, W. (1992). Dynamic programming algorithms for biological sequence comparison. *Methods in Enzymology*, *210*, 575-601.

Pennisi, E. (2010). Genomics. 1000 genomes project gives new map of genetic diversity. *Science*, *330*(6004), 574-5. doi: 10.1126/science.330.6004.574

Pratt, B., Howbert, J. J., Tasman, N. I., & Nilsson, E. J. (2012). Mr-tandem: parallel x!tandem using hadoop mapreduce on amazon web services. *Bioinformatics*, *28*(1), 136-137. doi: DOI10.1093/bioinformatics/btr615

Quail, M., Smith, M., Coupland, P., Otto, T., Harris, S., Connor, T., ... Gu, Y. (2012). A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers. *BMC genomics*, *13*(1), 1.

Rothberg, J., Merriman, B., & Higgs, G. (2012). Bioinformatics. introduction. *Yale J Biol Med*, *85*(3), 305-8.

Schäffer, A., Aravind, L., Madden, T., Shavirin, S., Spouge, J., Wolf, Y., ... Altschul, S. (2001). Improving the accuracy of psi-blast protein database searches with composition-based statistics and other refinements. *Nucleic acids research*, *29*(14), 2994–3005.

Schena, M., Shalon, D., Davis, R. W., & Brown, P. O. (1995). Quantitative monitoring of gene-expression patterns with a complementary-dna microarray. *Science*, *270*(5235), 467-470. doi: DOI10.1126/science.270.5235.467

Schikuta, E., & Wanek, H. (2001). Parallel i/o. *International Journal of High Performance Computing Applications*, *15*(2), 162-168. doi: Doi10.1177/109434200101500208

Schumacher, A., Pireddu, L., Niemenmaa, M., Kallio, A., Korpelainen, E., Zanetti, G., & Heljanko, K. (2014). Seqpig: simple and scalable scripting for large sequencing data sets in hadoop. *Bioinformatics*, *30*(1), 119-120. doi: DOI10.1093/bioinformatics/btt601

Shvachko, K., Kuang, H. R., Radia, S., & Chansler, R. (2010). The hadoop distributed file system. *2010 Ieee 26th Symposium on Mass Storage Systems and Technologies (Msst).*

Sugano, S. (2009). [introduction: next-generation dna sequencing and bioinformatics]. *Tanpakushitsu Kakusan Koso*, *54*(10), 1233-7.

Taylor, R. C. (2010). An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. *BMC Bioinformatics*, *11 Suppl 12*, S1. doi: 10.1186/1471-2105-11-S12-S1

Thomas, P., Wood, V., Mungall, C., Lewis, S., & Blake, J. (2012). On the use of gene ontology annotations to assess functional similarity among orthologs and paralogs: a short report. *PLoS Comput Biol*, *8*(2), e1002386.

Wang, D., & Springer, J. (Eds.). (2015, October). *Studies of usability and user experience situations in bioinformaties software.*

White, T. (2012). Hadoop the definitive guide [Book Section]. Yahoo Press.

Wright, P. C., Noirel, J., Ow, S. Y., & Fazeli, A. (2012). A review of current proteomics technologies with a survey on their widespread use in reproductive biology investigations. *Theriogenology*, *77*(4), 738-765. doi: DOI10.1016/j.theriogenology.2011.11.012

Xu, J. C., Stolk, J. A., Zhang, X. Q., Silva, S. J., Houghton, R. L., Matsumura, M., ... Reed, S. G. (2000). Identification of differentially expressed genes in human prostate cancer using subtraction and microarray. *Cancer Research*, *60*(6), 1677-1682.

Zhang, X. A., Asara, J. M., Adamec, J., Ouzzani, M., & Elmagarmid, A. K. (2005). Data pre-processing in liquid chromatography-mass spectrometry-based proteomics. *Bioinformatics*, *21*(21), 4054-4059. doi: DOI10.1093/bioinformatics/bti660

VITA

VITA

EDUCATION

Ph.D.: Computer and Information Technology (2013.01 - present). Focus: Data Science/ Big Data Analytics/ Bioinformatics. Purdue University, West Lafayette, IN, 47905

M.S.: Medicinal Chemistry (2009.09 2012.12). Focus: design, synthesis and evaluation of anticancer small molecules. University of Minnesota, Twin Cities, MN 55455

B.S.: Pharmaceutical Sciences (2003.09 2007.07) Peking University, Beijing, China

PUBLICATIONS

1. Wang D, Andrews TE, Harki DA Cell Surface Markers of Cancer Stem Cells: Diagnostic Macromolecules and Targets for Drug Delivery, Drug Deliv. Transl. Res., 2013, 3, 121-142.
2. Wang D, Li YH, Wang YP, Gao RM, Zhang LH, Ye XS Synthesis and in Vitro Anti-Hepatitis B Virus Activity of Six-membered Azanucleosides Analogues, Bioorg. Med. Chem., 2011, 19, 41-51.
3. Wang D, Wang N, Ye XS Recent advances on aza-nucleoside analogues Chinese Journal of Medicinal Chemistry, 2010, 20, 319-328.
4. Pang L, Wang D, Zhou J, Zhang LH, Ye XS Synthesis of neamine-derived pseudodisaccharides by stereo- and regio-selective functional group transformations Org. Biomol. Chem., 2009, 7, 4252-4266.

CONFERENCE PRESENTATIONS

1. Wang D, Springer J Studies of Usability and User Experience Situations in Bioinformaties Software Grace Hopper Conference, Houston, TX, Oct 14-16 2015.
2. Wang D, Rathe SK, Zohar S, Hexum JK, Largaespada DA, Harki DA Synthesis and Evaluation of Parthenolide analogues: Chemical Probes and Therapeutic Agents, 244th ACS National Meeting, Philadelphia, PA, Aug 18-23, 2012.
3. Wang D, Rathe SK, Zohar S, Hexum JK, Largaespada DA, Harki DA Synthesis and Biochemical Evaluation of Novel Parthenolide Analogues, 50th Annual MIKI Meeting, Iowa City, IW, Apr 13-15, 2012.
4. Wang D, Hexum JK, Meece FA, Harki DA Semisynthetic Parthenolide Analogues: Chemical Probes of Biological Function and New Anticancer Agents, 49th Annual MIKI Meeting, Lawrence, KS, Apr 8-10, 2011.
5. Harki DA, Wang D, Hexum JK, Meece FA Semisynthetic Parthenolide Analogues: Chemical Probes of Biological Function & New Anticancer Agents, Gordon Research Conference (Stem Cell & Cancer), Ventura, CA, Feb 20-25, 2011.

PATENT

1. Harki DA and Wang D (2011) Anti-cancer and anti-inflammatory Parthenolide compounds U.S. Patent NO. WO 2012145678 A1; issued Oct 26th 2012.

TEACHING EXPERIENCES

1. Teaching Assistant, Purdue University, 2015.01 - Present: Visual Programming (CNIT17500)
2. Teaching Assistant, Purdue University, 2014.09 2014.12: Database fundamentals (CNIT 27200)
3. Teaching Assistant, University of Minnesota, 2009.09 2009.12: Medicinal Agents III (PHAR 6156)
4. 2010.01 2010.05: Pharmaceutical Immunology and Biotechnology (PHAR 6159)